Universidad Carlos III de Madrid

Trabajo de Fin de Grado

Desarrollo y evaluación de servicios de comunicación en tiempo real basados en entidades hipervinculadas

Elías Solana Ortigosa

Tutor del Trabajo:
Dr. José Ignacio Moreno Novella

21 de Septiembre del 2017

Índice

Abstract **5**

Prefacio **7**

Capítulo 1: Introducción y objetivos **9**
1.1 - Introducción al trabajo 9
1.2 - Entorno de realización 9
1.3 - Objetivos del trabajo 10
1.4 - Fases de realización 11
1.5 - Estructura de la memoria 13

Capítulo 2: Estado del arte **15**
2.1 - reTHINK 15
2.1.1 - Introducción a reTHINK 15
2.1.2 - Objetivos del proyecto reTHINK 16
2.1.3 - Arquitectura reTHINK 17
2.2 - Tecnologías Web 26
2.2.1 - JavaScript 26
2.2.2 - JSON 26
2.2.3 - HTML 27
2.2.4 - CSS 27
2.2.5 - HTTP 28
2.3 - WebRTC 29
2.3.1 - Origen y propósito de WebRTC 29
2.3.2 - Arquitectura de WebRTC 30
2.3.3 - WebRTC API 30
2.4 - Otras tecnologías 32
2.4.1 - Npm 32
2.4.2 - GIT 33
2.4.3 - Docker 33

Capítulo 3: Arquitectura del desarrollo **35**
3.1 - Diseño de la interfaz gráfica 36
3.1.1 - Pantalla inicial 36
3.1.2 - Búsqueda y muestra de resultados 37
3.1.3 - Chat 38
3.1.4 - Videollamada 39
3.1.5 - Ventanas de invitación 40

3.1.6 - Sistema de pestañas 41
3.2 - Uso de elementos reTHINK 42
3.3 - Estructura de los archivos 43

Capítulo 4: Desarrollo previo **45**
4.1 - Hyperty Toolkit 45
4.2 - Slider Hyperty 48

Capítulo 5: Desarrollo y validación de la aplicación **57**
5.1 - Creación de la aplicación 57
5.1.1 - Configuración del entorno de trabajo 57
5.1.2 - Despliegue del Hyperty Runtime 58
5.1.3 - Sistema de pestañas 59
5.1.4 - Motor de búsqueda 64
5.2 - Creación de un servicio de comunicación para la aplicación . 70
5.2.1 - Diseño e implementación de la interfaz gráfica 71
5.2.2 - Implementación de los hypertys en la aplicación 76
5.3 - Desarrollo de un nuevo 'hyperty' para la aplicación 82
5.3.1 - Hyperty y controlador 83
5.3.2 - Uso y almacenamiento de los controladores 85
5.3.3 - Integración en la aplicación 87
5.4 - Validación 88

Capítulo 6: Conclusiones y trabajos futuros **91**
6.1 - Conclusiones del desarrollo previo 91
6.2 - Evaluación de la arquitectura reTHINK 91
6.3 - Conclusiones generales 95
6.4 - Impacto económico y social 96
6.5 - Trabajos futuros 97

Capítulo 7: Marco regulador **99**

Capítulo 8: Presupuesto **101**

Lista de figuras

1 Comparativa gráfica entre redes del tipo client-server y P2P. . 16
2 Capas del reTHINK framework. Fuente: reTHINK-project.eu 17
3 reTHINK Framework: Esquema general. Fuente: reTHINK . 18
4 Ejemplo de la estructura de la base de datos del catálogo. Fuente: reTHINK 20
5 Esquema de operación entre dispositivos a través de proto-stubs. Fuente: reTHINK 21
6 Esquema del funcionamiento del Hyperty Runtime. Fuente: reTHINK 22
7 Esquema del sistema de comunicación descentralizada. Fuente: reTHINK 23
8 Objetos de datos y su sincronización. Fuente: reTHINK . . . 24
9 Mecanismo de gestión de identidades. Fuente: reTHINK . . . 25
10 Ejemplo de cabecera de mensajes HTTP de solicitud y respuesta para un acceso a www.google.es 29
11 Modelo trapeziodal de WebRTC. Fuente: Real-Time Communication with WebRTC [14] 31
12 Línea de comandos para el arranque de docker-compose . . . 34
13 Esquema gráfico: Configuración de los contenedores 35
14 Diagrama de funcionamiento de la aplicación 37
15 Esquema de muestra de resultados 38
16 Modelo de ventana de chat que cumple las especificaciones . . 39
17 Modelo de ventana de videollamada que cumple las especificaciones 40
18 Esquema de muestra de resultados 41
19 Estructura de archivos de la aplicación 43
20 Clonación de repositorios para la instalación de la 'Hyperty Toolkit' 46
21 Arranque de la 'Hyperty Toolkit' 47
22 Selección de hypertys a ejecutar en la 'Hyperty Toolkit' . . . 47
23 Esquema de funcionamiento del hyperty Hello World y los archivos que lo componen 49
24 Captura de pantalla del hyperty Hello World Reporter 50
25 Deslizador del hyperty Slider 51
26 Mecanismo de sincronización del hyperty Slider 51
27 Esquema de funcionamiento del hyperty Slider y los archivos que lo componen 53
28 Estructura de archivos de la configuración Docker 58

29 Instalación de la Hyperty Runtime 59
30 Estructura de código del sistema de pestañas 61
31 Ejemplo de sistema de pestañas por colores 62
32 Ejemplo de resultados de búsqueda en formato JSON 65
33 Objeto JSON que muestra los hypertys de un resultado con información adicional 66
34 Barra de búsqueda 67
35 Proceso de búsqueda y muestra de resultados 68
36 Muestra de resultados 69
37 Muestra en detalle de un resultado de una búsqueda 70
38 Esquema de los elementos gráficos de chat y videollamada . . 72
39 Pantalla de chat 73
40 Pantalla de chat y videollamada 74
41 Funciones asociadas a la interfaz gráfica del archivo videochat.js 75
42 Estructura del código del hyperty de videollamada (DTWebRTC) 76
43 Carga del hyperty de videollamada en la Hyperty Runtime local 77
44 Lista de elementos EventListener (escuchadores) presentes en la aplicación 78
45 Lista de archivos relacionados con el servicio de chat 79
46 Carga del hyperty de chat en la Hyperty Runtime local . . . 80
47 Procedimiento de intercambio de mensajes 81
48 Lista de funciones relevantes en la función de chat (omitidas las relacionadas con la función de videollamada 82
49 Separación del hyperty de videollamada en hyperty y controlador 84
50 Estructura de código del hyperty de videollamada múltiple . 86

Abstract

The European project reTHINK aims to create an architecture that allows operators to provide deperimetrised services. This arquitecture would bridge the gap between operator-based and web-based services, improving inter-operability in the first category and user privacy and quality of service in the latter.

The purpose of this reasearch is to tackle the arquitecture of reTHINK project and its characteristics and how they solve issues in the previously mentioned areas. This is achieved by the development of a real-time communication application that uses reTHINK principles, evaluating its performances.

The results of this analysis show that, even though reTHINK project solves the technical challenges accordingly, the explotation of the project presents difficulties, especially in the broader market. This research advises to focus on specific markets related to public sector due to its proficiency in the area of privacy policy.

Prefacio

El Trabajo de Fin de Grado sobre el cual versa esta memoria tiene por objetivo el análisis de la arquitectura desarrollada por el proyecto europeo reTHINK a través del desarrollo de una aplicación de servicios de comunicación en tiempo real que emplea los elementos de dicha arquitectura.

El proyecto reTHINK es un esfuerzo conjunto por parte de diez empresas europeas de telecomunicaciones para crear una arquitectura de red que permita superar las diferencias entre servicios proporcionados por el operador de la red y servicios web del tipo OTT (on-the-top). Este proyecto está apoyado y financiado por la Comisión Europea, y busca el desarrollo de estos dos modelos de servicios hacia una tercera vía que mejore las características de éstos en campos como la inter-operabilidad entre proveedores o la gestión de privacidad de los usuarios.

Por lo general, esta memoria sigue el orden de realización de este trabajo: En primer lugar se hace una aproximación al tema a tratar y se definen los objetivos del Trabajo, describiendo los aspectos técnicos a evaluar en la arquitectura reTHINK.

Despúes se describen las distintas tecnologías utilizadas durante la realización, poniendo especial énfasis en el proyecto reTHINK, su arquitectura y sus distintos elementos. Esta fase de familiarización con las tecnologías empleadas ha sido de gran relevancia en cuanto a tiempo y dedicación. Todos los conceptos expuestos en esta sección son empleados a lo largo del Trabajo, así como las distintas tecnologías no relacionadas con reTHINK (principalmente, tecnologías web y de comunicación en tiempo real) que también se utilizan.

La parte central del trabajo trata sobre el diseño y el desarrollo de la aplicación que nos permite evaluar la arquitectura del proyecto reTHINK. Durante la fase de diseño, se pone especial énfasis en los aspectos gráficos de la aplicación, ya que la interfaz gráfica condiciona el desarrollo realizado. A partir de ahí se describe el desarrollo de la aplicación, separado en tres partes: la primera describe el desarrollo de las funciones más básicas de la aplicación (notor de búsqueda y sistema de pestañas), la segunda el desarrollo del servicio de comunicación entre usuarios, y la tercera la adaptación del código que permite la comunicación entre distintos usuarios al mismo tiempo.

Cabe reseñar que existe un desarrollo previo que, aunque no crea elementos utilizados en la aplicación, si sirve para el desarrollo de ésta.

Posteriormente, esta memoria contiene las conclusiones del desarrollo realizado desde distintos ámbitos. En los apsectos técnicos, la arquitectura reTHINK cumple sus especificaciones de manera satisfactoria, con ciertas particularidades en aspectos como el motor de búsqueda. Desde una perspectiva socio-económica, el proyecto reTHINK tiene un impacto limitado en los mercados más generalistas, ofreciéndose el sector público como una alternativa viable para su posterior explotación económica. La viabilidad de estas posibilidades en el mundo real son descritas en la sección de trabajos futuros.

Al final de la memoria se incluye una sección que describe el marco legal asociado a los contenidos, y un presupuesto de realización del Trabajo.

Capítulo 1: Introducción y objetivos

En este capítulo inicial de la memoria se introduce el tema a tratar y las razones que motivan la necesidad de este trabajo a través de una descripción del entorno de realización, así como los objetivos a conseguir del mismo. También se describen las fases de realización de este Trabajo de Fin de Grado, y en último lugar se halla una descripción de la estructura de esta memoria.

1.1 - Introducción al trabajo

El desarrollo de este Trabajo de Fin de Grado se realiza entre Octubre del año 2016 y Abril del año 2017, durante mi estancia en los laboratorios T-Labs de la empresa Deutsche Telekom AG en Berlín, Alemania. A lo largo de esos seis meses estuve involucrado en el proyecto colaborativo reTHINK, el cual pasaré a describir más adelante.

Mi labor en los laboratorios (y por extensión el contenido de este trabajo) era la creación de una aplicación web que usara y englobara servicios de comunicación en tiempo real previamente desarrollados por Deutsche Telekom usando los conceptos y las herramientas de reTHINK. Estos servicios de comunicación finalmente fueron videoconferencia, chat y un motor de búsqueda de usuarios de estos servicios.

La aplicación en el mundo real de este trabajo desarrollado es la realización de una plataforma web para grandes ciudades en la cual el flujo de comunicación se extiende de manera vertical (entre el proveedor y los usuarios) y horizontal (entre usuarios), tema que queda fuera del alcance de este Trabajo de Fin de Grado.

1.2 - Entorno de realización

Entre los servicios ofrecidos a través de Internet en la actualidad se pueden distinguir dos modelos de negocio: uno en el que el proveedor del servicio es también el proveedor de la red en la que los servicios trabajan, y otro llamado OTT (over-the-top), en el cual el proveedor del servicio usa la red pública de Internet y no tiene el control de la red sobre la que trabaja (ya que no es el proveedor de la red). [1]

Como ejemplos para cada uno de las categorías descritas, un servicio de

televisión bajo demanda (video-on-demand) ofrecido por el proveedor de red entraría dentro de la primera categoría, y un servicio como Skype o Facebook entraría dentro del modelo de negocio on-the-top.[1]

Cuando los servicios son ofrecidos por el proveedor de red, también llamado de aquí en adelante el operador, para asegurar inter-operabilidad entre servicios son necesarios acuerdos entre los proveedores con estándares bien definidos, lo cual supone una traba en la difusión de estos servicios, y además les impide competir a nivel global, especialmente a los proveedores más pequeños. Por otro lado, este tipo de servicios suelen gozar de buena fiabilidad en su calidad de servicio y en la seguridad de las identidades de los usuarios.

Los servicios de la categoría OTT (on-the-top) se caracterizan en la actualidad por llevar la iniciativa en la innovación de las comunicaciones, además de presentar modelos de negocios competitivos. Como aspectos negativos, los usuarios de estos servicios no pueden operar con usuarios de otros dominios, ni tampoco pueden usar su identidad en otro servicio distinto. Por último, presentan mayores problemas de privacidad (al usar la red pública) y la calidad de servicio no está asegurada (al no ser el proveedor de red el mismo que el de la aplicación).

El proyecto reTHINK pretende abordar esta división existente en los servicios ofrecidos en Internet mediante una nueva arquitectura, no basada en las redes de telecomunicaciones clásicas, sino basada en la web, que funciona mediante relaciones P2P de confianza entre aplicaciones.

Así, los operadores podrán ofrecer servicios basados en el contenido que puedan competir con las grandes empresas web (de la categoría OTT) como Google o Facebook. Esta nueva arquitectura pretende ofrecer un contenido global y contextual, asegurando la inter-operabilidad entre proveedores y la calidad del servicio, y con un sistema flexible de identidades que también garantice la privacidad.

1.3 - Objetivos del trabajo

El objetivo general de este Trabajo de Fin de Grado es la validación de la arquitectura reTHINK a través del desarrollo y evaluación de una aplicación que implemente servicios de comunicación en tiempo real basados en microservicios.

De manera más concreta, esta aplicación debe incluir el servicio de videoconferencia 'DTWebRTC', el motor de búsqueda 'Global Discovery' (ambos desarrollados por Deutsche Telekom) y el servicio de chat desarrollado de manera colaborativa en el proyecto reTHINK. Con esta aplicación se debe poder hacer una búsqueda de usuarios, contactar con un usuario conectado, establecer una videollamada y un chat y liberar los recursos usados cuando la conexión haya terminado.

Además, se deberá poder mantener varias videoconferencias y chats con distintos usuarios al mismo tiempo, lo cual requerirá la creación de un nuevo microservicio capaz de proporcionarlo.

Los conceptos de reTHINK a evaluar son los siguientes, ordenados de mayor a menor por orden de relevancia:

- Reusabilidad de hypertys (entidades hipervinculadas) como microservicios
- Sincronización: Mecanismo de sincronización de los objetos de datos
- Motor de búsqueda: Funcionamiento y practicidad del motor de búsqueda 'Global Discovery'
- Protocolo 'on-the-fly'
- Gestión de identidades: Independencia de las entidades de los usuarios respecto a los proveedores de servicios

1.4 - Fases de realización

En la siguiente lista se detallan los siguientes pasos seguidos durante la realización del Trabajo:

- **Introducción a la materia**
  Planteamiento del problema y aprendizaje previo de las tecnologías existentes relativas al proyecto (en especial Javascript) y del proyecto reTHINK.

- **Desarrollo previo: crear un hyperty bi-direccional**
  Crear un nuevo hyperty para comprobar la posibilidad de que sea bi-direccional y un mecanismo de sincronización que cumpla la condición.

- **Diseño de la aplicación**
  Definir los componentes de la interfaz gráfica necesarios para la aplicación, y cómo interactúan entre sí, definiendo los pasos del funcionamiento.

- **Crear la aplicación**
  Crear una aplicación web que use el Hyperty Runtime (sistema en tiempo de ejecución para hypertys) y sea capaz de desplegar un hyperty y sus funcionalidades asociadas (catálogo, registro)

- **Crear un servicio de comunicación para la aplicación**
  Crear un servicio de comunicación basado en hypertys (microservicios) ya desarrollados y desplegarlo en nuestra aplicación.

- **Desarrollar un 'hyperty' (microservicio) para la aplicación**
  Crear un hyperty capaz de mantener varias videoconferencias y chats al mismo tiempo con distintos usuarios y desplegarlo en nuestra aplicación.

- **Prueba y evaluación de la aplicación desarrollada y sus servicios**
  Prueba y 'debugging' de todas las funciones y escenarios de la aplicación y sus servicios.

- **Conclusiones**
  Conclusiones y feedback del proyecto reTHINK, la aplicación y sus futuras aplicaciones.

- **Documentación del trabajo**
  Creación de videos de demostración del funcionamiento de la aplicación, toma de capturas de pantalla y redacción de la memoria del trabajo.

Al final de esta memoria se encuentra un diagrama de Gantt que incluye todas las fases antes descritas, divididas en sub-tareas y asignadas a una franja de tiempo determinada.

1.5 - Estructura de la memoria

La memoria se divide en 8 capítulos, los cuales se agrupan en 3 bloques distintos.

La primera parte sirve como introducción. En el capítulo 1 se introduce el tema del Trabajo de Fin de Grado, su entorno de realización, sus objetivos, sus fases de realización e información relacionada con su memoria. En el capítulo 2, denominado Estado del Arte, se realiza en primer lugar un repaso al proyecto reTHINK, su arquitectura y las distintas herramientas desarrolladas en el momento de realización del Trabajo. Después se hace un repaso a la distintas tecnologías existentes utilizadas en la realización de este Trabajo.

La segunda parte describe el diseño y el desarrollo realizados y su validación. En el capítulo 3 se describe el diseño concebido para la aplicación, teniendo en cuenta las especificaciones descritas en el primer capítulo. El capítulo 4 describe el desarrollo previo realizado que, si bien no forma parte del desarrollo de la aplicación, si influye en la elaboración de ésta. En el capítulo 5 se detalla en tres pasos el desarrollo de la aplicación, y por último el método empleado para validar su funcionamiento y los resultados de ésta validación.

La tercera y última parte de la memoria incluye las conclusiones del Trabajo y varios aspectos relacionados. En el capítulo 6 se incluyen las conclusiones de este Trabajo, su impacto socio-económico y los desarrollos futuros. El capítulo 7 enumera las leyes y disposiciones legales que afectan al desarrollo realizado en este Trabajo. Por último, el capítulo 8 incluye un presupuesto de la realización de este Trabajo.

Al principio de la memoria se incluye un índice detallado de las secciones de ésta y un índice de las figuras, encontrándose la bilbiografía empleada al final.

Capítulo 2: Estado del arte

En esta capítulo de la memoria se enumeran y describen de manera general las distintas tecnologías existentes en el momento de la realización del trabajo.

Este capítulo se divide en 4 partes: La primera es una descripción del proyecto reTHINK y su arquitectura, poniendo énfasis en los conceptos más relevantes para este trabajo. La segunda contiene las teconlogías usadas en la World Wide Web que han sido empleadas en la elaboración de este trabajo. La tercera sección incluye los protocolos de comunicación usados en las comunicaciones en tiempo real en la Web. La cuarta y última sección es un compendio de tecnologías empleadas que no entran en ninguna de las otras categorías definidas.

2.1 - reTHINK

2.1.1 - Introducción a reTHINK

reTHINK es un proyecto de investigación europeo cuyo principal objetivo es diseñar y crear un prototipo de una nueva arquitectura de servicio P2P centrada en la web, haciendo posible relaciones de confianza entre aplicaciones distribuidas. Estas relaciones se denominan Hyperlinked Entities ("Hyperties"), y soportan usos de comuicación social o usos del tipo M2M/IoT, entre otros. Este projecto permite la entrega de cualquier tipo de servicio con especificaciones de calidad de red, impulsado por servicios especializados P2P o Cloud Computing. [2]

Las empresas e instituciones asociadas al proyecto reTHINK son Eurescom GmbH, Orange SA, Deutsche Telekom AG, Portugal Telecom Inovação e Sistemas SA, Fraunhofer Gesellschaft, Apizee, Institut Mines-Telecom, INESC ID, Quobis Networs SL y Technische Universität Berlin. [3]

El proyecto reTHINK está subvencionado por la Unión Europea dentro del programa Horizon 2020. Este programa es el instrumento de financión de la iniciativa 'Innovation Union'. cuyo objetivo es asegurar la competitividad global de Europa. [4]

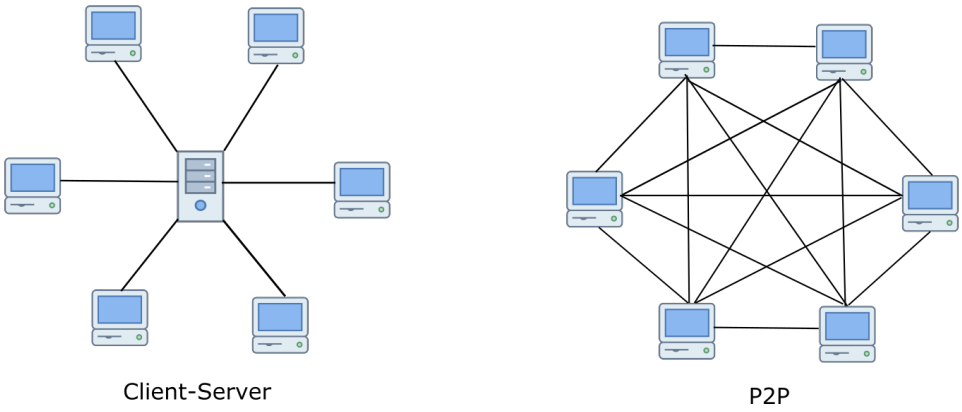


Figure 1: Comparativa gráfica entre redes del tipo client-server y P2P.

2.1.2 - Objetivos del proyecto reTHINK

Para cumplir la misión espuesta con anterioridad de desarrollar una nueva arquitectura de servicio P2P y las funcionalidades asociadas para los hypertys, reTHINK se propone 5 objetivos a conseguir:

- **Objetivo 1**

  Proveer un marco de trabajo (framework) de comunicación que facilite y soporte comunicaciones entre usuarios. Este reTHINK framework consistirá de cuatro capas: La Capa Hyperty, la Capa de Mensaje, la Capa de Transporte y una Capa Gestor de Confianza (Trust Management) transversal a las 3 anteriores (mostrado en la figura 2).

- **Objetivo 2**

  Diseñar y desarrollar las funcionalidades de seguridad y portabilidad necesarias para asegurar un servicio de fiable y de confianza, compatible con las regulaciones nacionales y la europea. Además, proponer soluciones para medir el nivel de confianza de las identidades ofrecidas por una tercera entidad.

- **Objetivo 3**

  Examinar el impacto de negocio de los conceptos desarrollados por el proyecto reTHINK en las comunicaciones y los servicios actuales, y como podrían transformar el futuro. Este objetivo también incluye analizar el impacto que reTHINK pueda tener en la eonomía y la sociedad en general, especialmente en la europea.

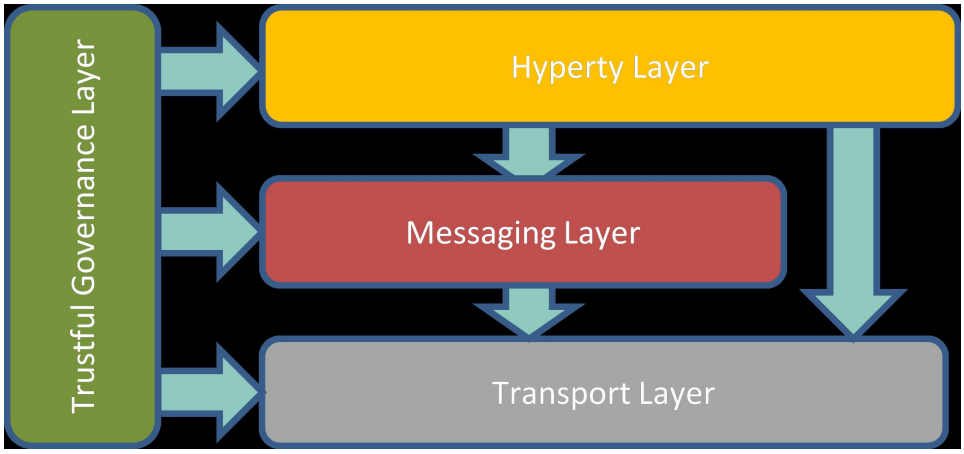


Figure 2: Capas del reTHINK framework. Fuente: reTHINK-project.eu

- **Objetivo 4**

  Validar la efectividad de reTHINK aplicándolo a una serie de casos que representen las áreas de aplicacioón más importantes, como por ejemplo comunicación de vídeo en tiempo real o comunicación M2M a través de IoT. Las áreas de aplicación también incluyen servicios de telecomunicación clásicos, domótica o entretenimiento.

- **Objetivo 5**

  Efectuar las actividades de estandarización y explotación, así como la diseminación de los resultados para incrementar la competitividad europea en el campo de las comunicaciones en tiempo real en Internet.

Debido a los contenidos de y los objetivos de este Trabajo de Fin de Grado, la mayor parte del trabajo está directamente relacionado con el objetivo 4, aunque las conclusiones de éste están también relacionadas con el objetivo 3.

2.1.3 - Arquitectura reTHINK

Como se ha descrito en los apartados anteriores, el principal objetivo del proyecto reTHINK es el diseño y creación de una arquitectura con unas determinadas características. En esta sección pasaremos a describir esa arquitectura, sus funcionalidades y más en detalle las áreas relevantes para este Trabajo de Fin de Grado.

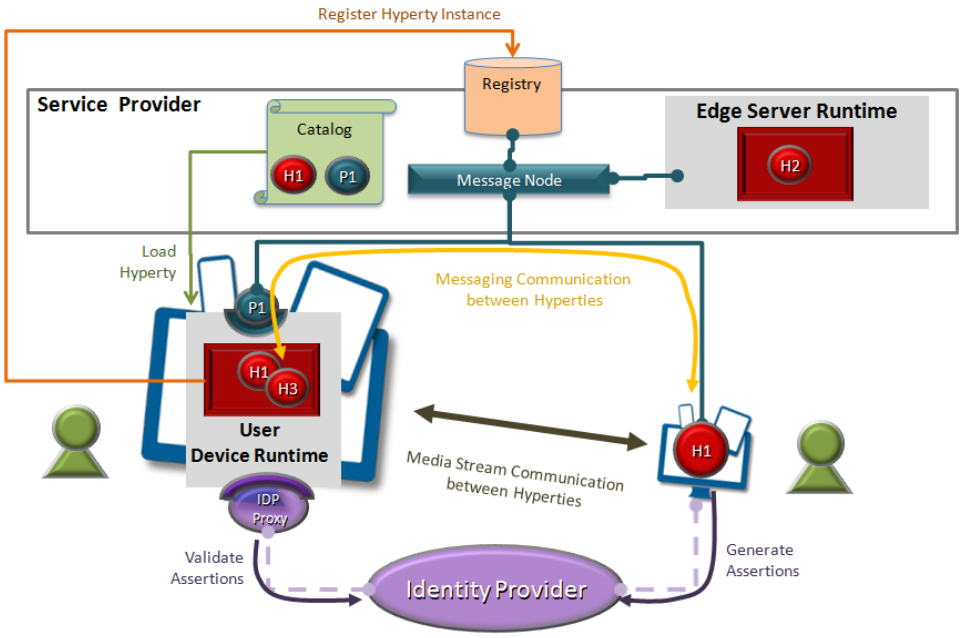


Figure 3: reTHINK Framework: Esquema general. Fuente: reTHINK

Arquitectura general

La arquitectura reTHINK ofrece las herramientas para construir una red global descentralizada de hypertys que son ejecutados en la capa más elevado y se comunica de manera segura a través de un sistema de comunicación descentralizada.

En la figura 2 se pueden distinguir de manera esquemática los elementos principales de la arquitectura reTHINK. El primer elemento es el proveedor de servicio (Service Provider), el cual engloba el catálogo (Catalogue), el registro (Registry) y el Message Node, los cuales son objeto de estudio para nosotros. El segundo es el Runtime de usuario o Hyperty Runtime (User Device Runtime en la figura). Como último elemento estudiaremos el proveedor de identidades (Identity Provider) y cómo interactúa con el Hyperty Runtime.

A lo largo de los siguientes apartados pasamos a definir y conocer en más detalle cuales son los procedimientos de comunicación usados en el sistema.

Pero antes pasaremos a definir los elementos más relevantes con los que se forma ésta arquitectura.

Hyperty Hyperty, semánticamente derivado de la contracción de las palabras Hyperlink (hipervínculo) e Identity (identidad), son módulos de software que siguen el patrón de arquitectura de microservicios , lo que significa que son componentes que se despliegan de manera independiente ofreciendo cada uno una serie de capacidades de uso. [5]

Por ejemplo, un hyperty puede ofrecer un microservicio de chat, y otro hyperty puede ofrecer un microservicio de envío de imágenes, creando un servicio de mensajería entre usuarios. Asḿismo, se podría crear un servicio de videollamada y chat similar a Skype usando un hyperty que ofrezca comunicación de vídeo y el mismo hyperty que ofrezca un chat mencionado anteriormente. En reTHINK son los objetos en torno a los que gira toda la arquitectura.

Los hypertys están programados en el lenguaje JavaScript (aunque en el futuro se espera ampliar el número de lenguajes), y se basan en un método sencillo y ligero de comunicación descentralizada que mantienen entre sí. Esto proporciona las siguientes ventajas:

Son inter-operables: Los hyperties se comunican entre si de manera descentralizada usando los conceptos Protocolo on-the-fly (descrito más adelante) y el esquema de comunicación Reporter - Observer para ofrecer interoperabilidad entre distintos dominios. En un caso práctico de funcionamiento, un hyperty asociado a un sensor (con el rol Reporter) enviaría un objeto del tipo JSON con la información relevante al hyperty de escucha (con el rol Observer) en cuanto lea una nueva medida en el sensor. De este modo, los objetos JSON a ambos lados están siempre sincronizados.

Son fiables: Los hypertys usan un sistema de gestión de identidades independiente del proveedor de servicios y de la aplicación. Esta identidad es gestionada por el usuario y verificada por una entidad independiente seleccionada por el mismo.

Son reutilizables: El desarrollo de hypertys es sencillo y flexible. Actualmente los hypertys están programados en Javascript, lo que permite que

Figure 4: Ejemplo de la estructura de la base de datos del catálogo. Fuente: reTHINK

cualquier dispositivo con un navegador web pueda ejecutarlos sin ninguna instalación adicional

Catálogo El catálogo de reTHINK (referido en otras partes del texto como Catalogue) es una base de datos de objetos necesarios para el funcionamiento de la arquitectura reTHINK. Los objetos que almacena el catálogo pueden ser de los siguientes tipos: protocolstub, hyperty, dataschema, runtime e idp-proxy. En la figura 3 se puede apreciar la estructura que sigue la base de datos del catálogo.

Desde el punto de vista del usuario, se pueden realizar dos acciones con el catálogo: conseguir información de un objeto, y actualizar la información de los objetos, accediendo a través de los puertos 80 (HTTP) y 443 (HTTPS).

Protostub y Protocol on-the-fly Protostub, de la contracción de Protocol Stub, es una pieza de código que consigue que el navegador de un dispositivo entienda el protocolo de un Message Node de otro proveedor.

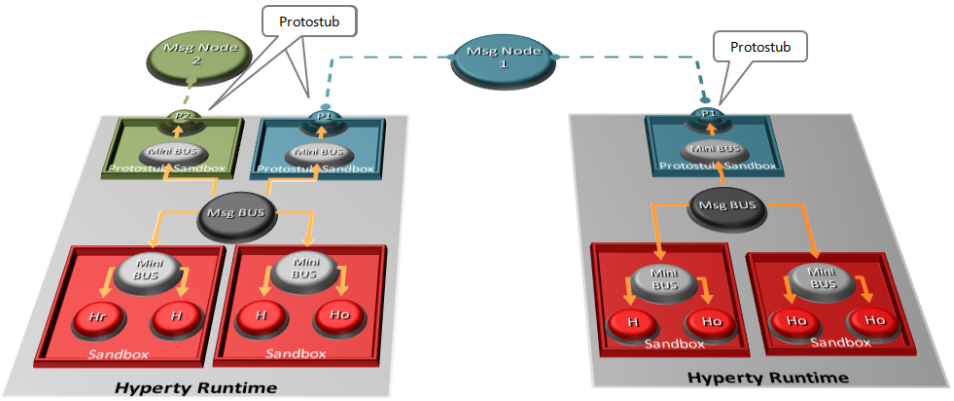


Figure 5: Esquema de operación entre dispositivos a través de protostubs. Fuente: reTHINK

Antes de establecer la conexión entre usuarios, el dispositivo carga el protostub correspondiente (descargado desde el catálogo), al igual que el dispositivo al otro lado. Una vez cargados, estos dispositivos pueden comunicarse directamente.

La filosofía de trabajo entre protostubs está definida por el concepto 'Protocol on-the-fly'. Éste consiste en la selección dinámica (durante el tiempo de ejecución) y carga del protocolo más apropiado para la comunicación. Esto permite la inter-operabilidad entre servicios distribuidos, evitando la necesidad de tener protocolos de entrada en la red que conecta los distintos servicios.

Hyperty Runtime Hyperty Runtime (en la figura 2 descrito como User Device Runtime) es el sistema en tiempo de ejecución que, instalado en cada dispositivo, permite la operabilidad con otros dispositivos usando hypertys. Éste código tiene distintos módulos, los cuales interactúan entre sí y con el exterior, como se puede ver en la figura 6.

Desde el Hyperty Runtime, el acceso a cada uno de los proveedores de servicio se produce a través de su correspondiente protostub (como se indica en la sección previa dedicada a ello), el cual internamente se conecta al Message Bus desplegado por el Core Runtime del proveedor de servicios, y de ahí al hyperty. De modo inversa, el hyperty se conecta al Message Bus,

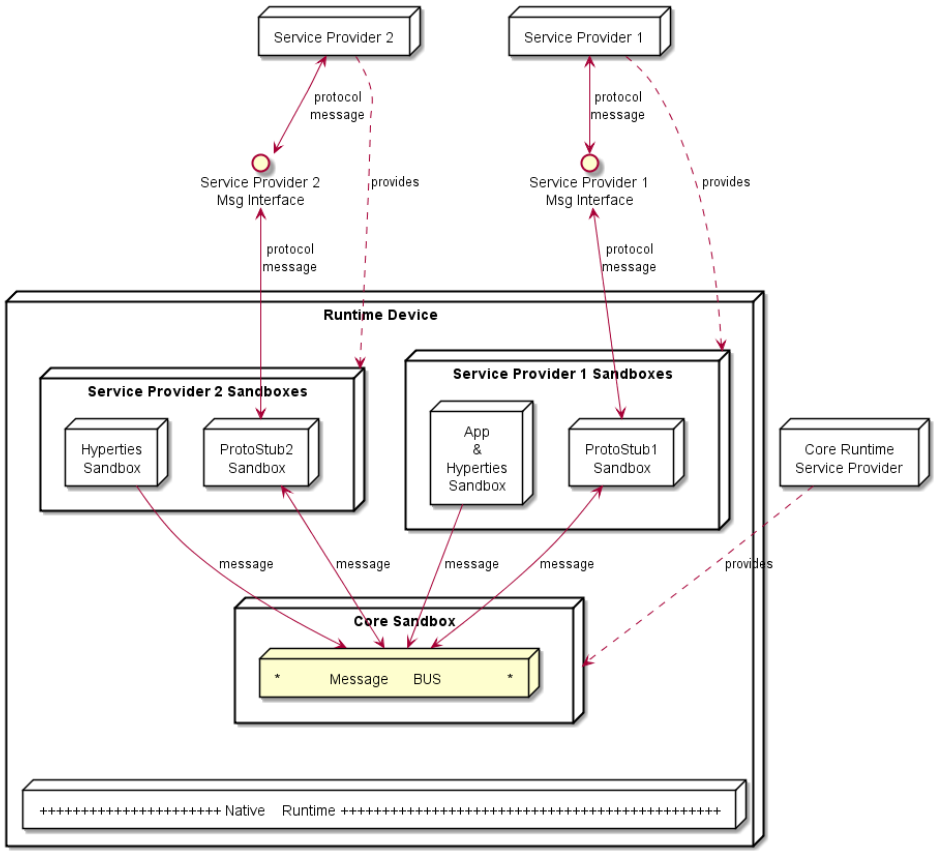


Figure 6: Esquema del funcionamiento del Hyperty Runtime. Fuente: re-THINK

y esta transfiere la información al protostub que le corresponde, para luego salir al exterior. Este proceso queda detallado en la figura 6.

En el interior del Hyperty Runtime, los hypertys de proveedores de servicios distintos se despliegan en 'sandboxes' (módulos) distintos, por motivos de seguridad. Asimismo, los protostubs y los hypertys se despliegan en 'sandboxes' separados entre sí, aunque pertenezcan al mismo proveedor de servicios, por la misma razón.

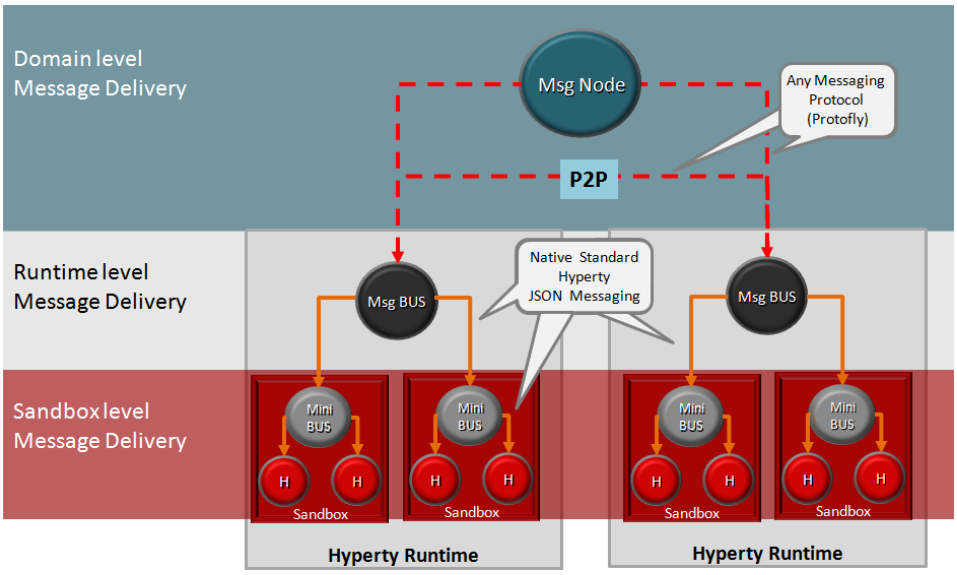


Figure 7: Esquema del sistema de comunicación descentralizada. Fuente: reTHINK

Modelo de Comunicación Descentralizada El objeto de datos es la unidad básica de comunicación en reTHINK, y son los mensajes que se intercambian los hyperties de cada usuario final conectados entre sí. Estos mensajes están escritos en el formato de texto JSON, y están pensados para realizar operaciones CRUD (crear, leer, actualizar y borrar). Estas operaciones las realizan sobre los recursos mantenidos en los destinatarios finales (end-points). Un ejemplo de operación sería la creación o actualizción del estado de una videollamada.

Cada mensaje tiene una cabecera para permitir su enrutamiento, que incluye los campos de id, from (dirección de la entidad de procedencia), to (dirección del destinatario) y type(identifica la operación a realizar). El cuerpo del mensaje se compone de distintos tipos de información que dependen de la operación CRUD a realizar.

El envío de estos mensajes se basa en una sencilla funcionalidad de enrutamiento que ejecuta un 'look-up' en busca de sistemas registrados a la escucha. El mensaje se publica para todos los que escuchan, sean routers o destinatarios finales, de tal forma que cada router sólo conoce routers o destinatarios finales adyacentes. En la figura 7, se puede apreciar como el

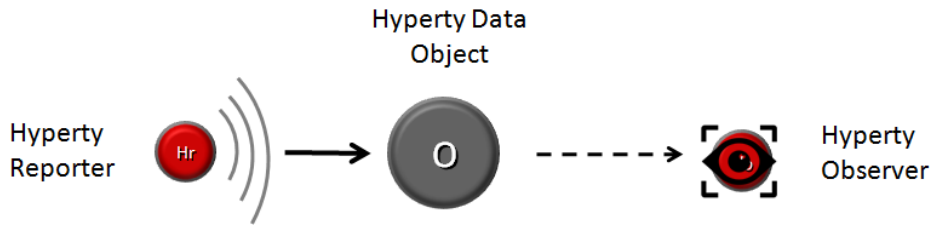


Figure 8: Objetos de datos y su sincronización. Fuente: reTHINK

papel de routers lo protagonizan los 'Mini Bus', los 'Message Bus' y el 'Message Node'.

Sincronización de datos Los mensajes descritos anteriormente se envían siguiendo un esquema de sincronización de datos P2P: En una conexión entre dos hypertys, uno de los hypertys toma el rol de 'Reporter', y otro el de 'Observer'. De este modo, para evitar problemas de concurrencia entre los sistemas conectados, sólo uno puede tomar permisos para escribir datos en el objeto de datos (el mensaje), mientras el otro (o más) sistema sólo puede leer el objeto.

Tan pronto como el 'Reporter' realiza un cambio en el objeto de datos, este cambio se propaga inmediatamente a través del sistema de comunicacón descentralizada descrito en la sección anterior. Con este sistema de sincronización automática, el objeto de datos manejado por el 'Observer' tiene el mismo contenido que el manejado por el 'Reporter'.

De este modo se consigue uno de los objetivos principales del proyecto reTHINK: dos sistemas son completamente inter-operables necesitando sólo ponerse de acuerdo en el formato del objeto de datos.

Gestión de identidades La gestión de identidades de reTHINK busca la flexibilidad en beneficio del usuario, manteniendo los niveles de seguridad necesarios. Cada hyperty está asociado a una identidad de usuario elegida por el usuario mismo. Éstos usuarios pueden ser humanos, empresas u objetos (por ejemplo, un sensor). En reTHINK, las identidades no están limitadas a un único proveedor, por lo que el usuario puede elegir el provee-

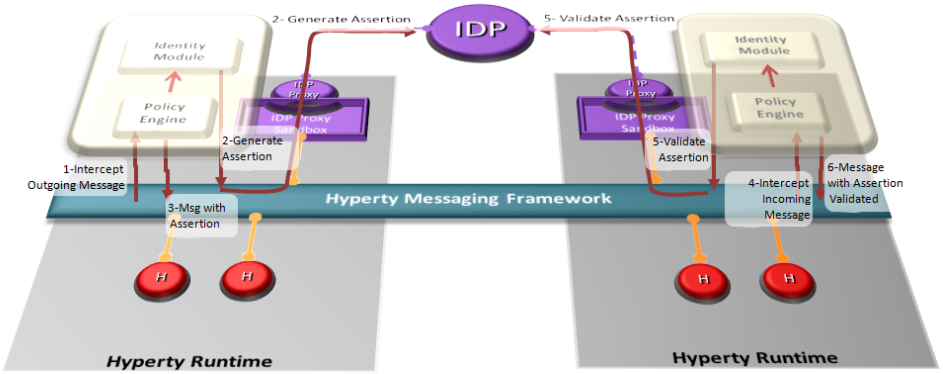


Figure 9: Mecanismo de gestión de identidades. Fuente: reTHINK

dor de identidades que mas desee.

En la figura 9 se puede ver el mecanismo completo de gestión de identidades y los elementos que lo forman. El Identity Module (módulo de identidad) es el componente en el 'Runtime' de manejar la identidad del usuario y la asociación de ésta a los distintos hypertys. Los Identity Provider Proxys (IdP Proxy en la figura) actúan de manera análoga a los protostubs, siendo descargados de los proveedores de identidades y tomando la responsabilidad de la comunicación con éstos. El tercer elemento de relevancia en esta arquitectura es el 'Policy Engine', encargado de administrar y aplicar polt́icas de acceso.

En reTHINK el proceso de autentificación es mutuo. Como se puede apreciar en la sucesión de acciones descrita en la figura 9, cada vez que un hyperty intenta enviar un mensaje, se genera una 'identity assertion' (afirmación de identidad), que a través del sistema descrito el en párrafo anterior verifica que el mensaje enviado está asociado a una identidad válida. Cuando esa identidad está validad y ese mensaje llega al destinatario, este nuevo sistema repite el proceso anterior. Además, este sistema de verificación mutua es esencial para la comprobación de las claves de cifrado simétricas usadas en las comunicaciones seguras.

2.2 - Tecnologías Web

2.2.1 - JavaScript

JavaScript es un lenguaje de programación de alto nivel, dinámico y orientado a objetos, usado junto a HTML y CSS para producir aplicaciones web. [6]

JavaScript es uno de los lenguajes de programación más usados del mundo, siendo el más popular en la plataforma GitHub [6]. Sólo es necesario un navegador web para poder usar JavaScript, ya que los navegadores web más populares (Chrome, Firefox, Internet Explorer) incluyen motores Javascript, aunque no es necesario un navegador web para ejecutar JavaScript.

La decisión de hacer de JavaScript ES6 el lenguaje de programación oficial del proyecto reTHINK fue tomada al principio de éste. Las principales ventajas tenidas en cuenta fueron su popularidad y su capacidad de programación orientada a objetos. Además, el formato JSON (JavaScript Object Notation) tiene una gran compatibilidad con ciertas funciones del proyecto reTHINK. En cualquier caso, no todas las funciones de reTHINK han sido creadas en JavaScript (por ejemplo, el sistema Global Discovery fue programado en Java).

En la elaboración de este Trabajo de Fin de Grado no se ha usado ninguna librería de JavaScript.

2.2.2 - JSON

El formato de datos JavaScript Object Notation (descrito a lo largo de toda la memoria por su acrónimo, JSON) se deriva del lenguaje de programación JavaScript, y sirve como formato de intercambio de datos. El primer objeto JSON conocido data del año 2006. [7]

La sintaxis de JSON destaca por su sencillez: Los datos JSON están escritos en pares de nombre y valor. El nombre va entrecomillado (a diferencia de JavaScript), y los valores que soporta son de los siguientes tipos: string, número, objeto (JSON), array, boolean y null.

2.2.3 - HTML

HTML (del acrónimo en inglés Hypertext Markup Language) es el lenguaje de marcado estándar para crear páginas web. 'Hypertext' se refiere al texto presentado en un ordenador que contiene enlaces para acceder a otros documentos 'Hypertext'. Un documento 'hypertext' describe objetos tales como listas, tablas o imágenes. [8]

La versión actual de HTML es HTML5, versión que soportan los navegadores web más populares (Chrome, Internet Explorer, Firefox, Safari y Opera).[8]

En este trabajo hemos usado HTML ya que es el estándar para aplicaciones web. Aunque en la fase inicial del trabajo se iba a usar en teoría un 'mock-up' creado previamente para el proyecto reTHINK, éste 'mock-up' no llegó a tiempo, por lo que todo el código HTML está implementado desde la base, atendiendo a las distintas funcionalidades de la aplicación.

A lo largo del trabajo se emplean distintas funcionalidades de HTML5, como por ejemplo video, al margen de las funcionalidades más básicas. Además, debido a que la aplicación requiere funcionalidades dinámicas (no todos los elementos HTML se cargan desde el principio de la ejecución), el acceso a los elemntos HTML desde el código JavaScript es muy común a lo largo de todo el código fuente, lo cual complica su lectura y posterior 'debugging'.

2.2.4 - CSS

CSS es la abreviatura del inglés de Hoja de Estilos en Cascada (Cascading Style Sheets). Es un lenguaje que sirve para dar formato a lenguajes de marcado (HTML en nuestro caso de trabajo), permitiendo separar el contenido del documento y la presentación. La versión actual de CSS es CSS 3, que consiste en varios documentos llamados mdoulos. [8]

En la elaboración de este trabajo hemos usado CSS ya que, aunque no afectaba a la estricta funcionalidad de la aplicación (como sí lo hacen HTML y JavaScript), sí que era conveniente para la correcta visualización de ciertas funcionalidades de la aplicación, como pueden ser el motor de búsqueda o la ventana de chat.

Todos los elementos CSS empleados a lo largo del trabajo han sido creados desde la base, sin usar ninguna plantilla. Además, de manera análoga a lo expuesto en el apartado anterior, debido a la naturaleza dinámica de la aplicación, ha sido necesario acceder a ciertos elementos CSS desde el código JavaScript.

2.2.5 - HTTP

El término HTTP (Hypertext Transfer Protocol) corresponde al protocolo a nivel de aplicación para sistemas distribuidos, colaborativos y de información hipermedia que lleva dicho nombre. Además de su uso para hipertexto, extendiendo sus capacidades (de los métodos request) puede efectuar varias tareas más, como por ejemplo nombrar servidores. En la web global (World-Wide Web) lleva en uso desde el año 1990. [9]

El protocolo HTTP se basa en el sistema de solicitud y respuesta (request/response), y se aplica a redes del tipo cliente-servidor. De esta manera, un cliente (un usuario en la mayoría de los casos) manda una solicitud a un determinado servidor, y este servidor devuelve la respuesta con los datos que el cliente había solicitado.

Los métodos HTTP indican la acción a realizar en el recurso alojado en el servidor al que queremos acceder. Lós métodos definidos por el estándar HTTP/1.1 son GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE y CONNECT. Aunque en el proyecto reTHINK se hace un uso extensivo de los distintos métodos de HTTP, en la aplicación de la que es objeto este Trabajo de Fin de Grado se usa solamente el étodo GET (lo cual veremos en la sección Desarrollo y Validación). El método GET se usa únicamente para extraer datos, y no realiza ninguna otra acción

Además de los métodos, HTTP incorpora 'status codes' (códigos de estado), que dan información al cliente del proceso realizado en el servidor, así como su estado. Éstos códigos tienen tres cifras, siendo la primera la más relevante. [9]

- Código tipo 1xx: Indica que la solicitud ha sido recibida y entendida
- Código tipo 2xx: Indica que además de lo anterior, ha sido procesada con éxito

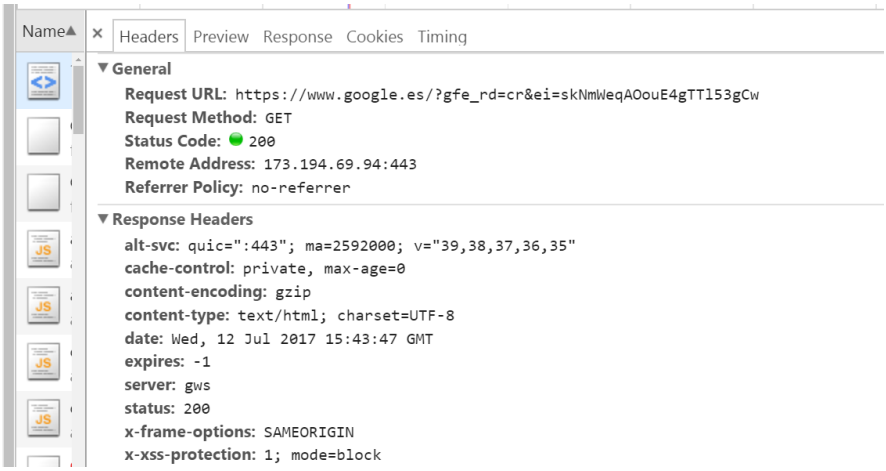


Figure 10: Ejemplo de cabecera de mensajes HTTP de solicitud y respuesta para un acceso a www.google.es

- Código tipo 3xx: Indica que el cliente debe realizar alguna acción adicional
- Código tipo 4xx: Indica un error en la solicitud
- Código tipo 5xx: Indica un error en el servidor (la solicitud es correcta)

Cómo último concepto de interés para un entendimiento básico del protocolo HTTP, definiremos qué es una URI. URI, acrónimo del inglés de Uniform Resource Identifier, es una cadena de texto que identifica un recurso de la red de manera unívoca. Hay dos tipos: URN (Uniform Resource Name) y URL (Uniform Resource Location), siendo más popular éste último.

2.3 - WebRTC

Web Real-Time Communication (WebRTC) es un estándar que define el intercambio directo entre navegadores de contenido multimedia en tiempo real.

2.3.1 - Origen y propósito de WebRTC

En la actualidad, existe una extensa implantación de la comunicación por vídeo, con servicios como Google Hangouts, Skype o Facetime. Sin embargo, muchos servicios de videollamada requieren la instalación de aplicaciones o

plug-ins adicionales por parte del usuario, así como de pago de licencias por parte de los desarrolladores.

El propósito de WebRTC es implantar la tecnología de comunicación de video entre usuarios en los navegadores, haciendo posible la implantación de esta tecnologá sin la necesidad de plug-ins u otros requisitos, pudiendo establecer una conexión de buena calidad entre los usuarios directamente. [10]

Como punto de partida, en el año 2011 Google lanzó el proyecto de software libre WebRTC, con los propósitos indicados en el párrafo anterior y el objetivo de implantarlo en el navegador Google Chrome, con la previsión de una futura implantación en el resto de navegadores [11]

Desde entonces, el organismo IETF(Internet Engineering Task Force) ha trabajado en la estandarizaicón de los protocolos [12], y W3C (World Wide Web Consortium) ha trabajado en la estandarización de las APIs [13]

2.3.2 - Arquitectura de WebRTC

En el modelo general (trapezoidal) mostrado en la figura 11, los dos navegadores ejecutan una aplicación web descargada de un servidor web. Cabe notar que, aunque en la figura se muestra un caso general para el que cada usuario descarga la aplicación de un servidor distinto, habitualmente los usuarios descargan la aplicación del mismo servidor, por lo que el modelo de la figura tendría una forma triangular.

La comunicación entre el servidor web y el navegador sirve para establecer y terminar la comunicación, a través del protocolo HTTP o WebSockets, pero este tramo de la comunicación no está definida por WebRTC (ya que es parte de la aplicación). Para la comunicación de datos multimedia se utiliza la API PeerConnection, que conecta a los navegadores sin necesidad de ningún servidor. [14]

2.3.3 - WebRTC API

La API WebRTC 1.0 definida por el World Wide Web Consortium [13] permite desarrollar la tecnología WebRTC en un entorno JavaScript, y gira en torno a tres conceptos principales: MediaStream, RTCPeerConnection y DataChannel. [14]

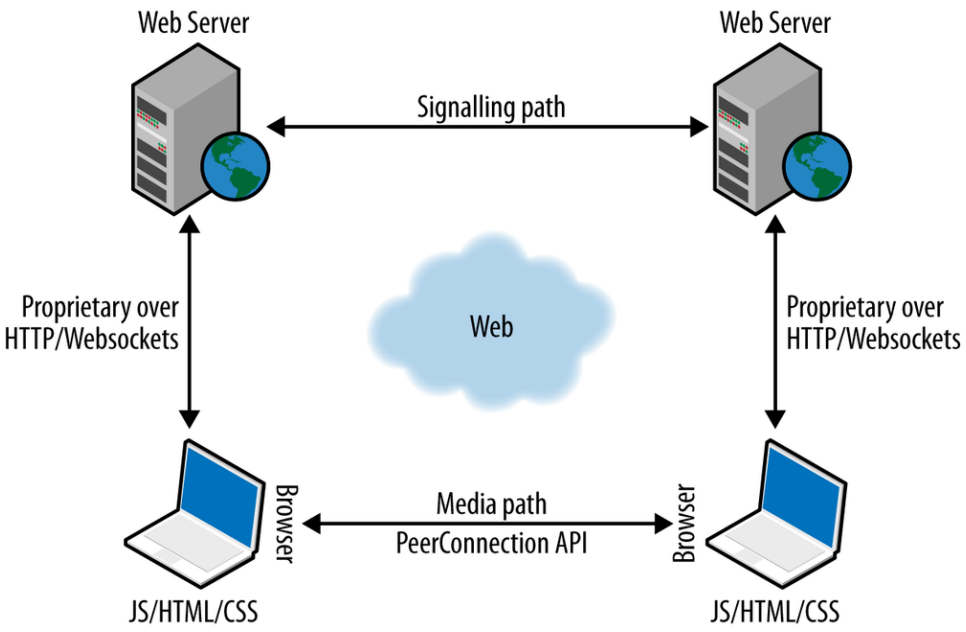


Figure 11: Modelo trapeziodal de WebRTC. Fuente: Real-Time Communication with WebRTC [14]

La interfaz MediaStream representa un flujo o corriente (traducido del inglés stream) de datos multimedia. Un objeto MediaStream puede representar el flujo procedente de una cámara local (LocalMediaStream) o del otro usuario (Remote). Cada interfaz MediaStream contiene varias pistas (MediaStreamTrack) que representa un flujo o corriente de video, o de audio, por ejemplo. Cada objeto MediaStream tiene un identificador único, y sus propiedades son definidas por el objeto MediaStreamConstrains. [13]

Una instancia RTCPeerConnection permite establecer comunicación directa desde un navegador con otra instancia RTCPeerConnection en otro navegador, u otro dispositivo que implemente los protocolos necesarios. Por otro lado, la instancia RTCConfiguration define una serie de parámetros que configuran como se establece la comunicación a través de RTCPeerConnection. RTCPeerConnection se vale del protocolo Interactive Connectivity Establishment (ICE) para permitir la conexión en redes con direcciones IP modificadas por el protocolo NAT (Network Address Translation). [15]

La interfaz RTCDataChannel representa un canal de datos bi-direccional entre dos usuarios, y se crea cuando la función CreateDataChannel sobre un objeto RTCPeerConnection. La instancia RTCDataCHannel esta asociado a un canal de transporte de datos que envía datos de forma efectiva al otro usuario. Las características de este canal de transporte se configuran cuando la instancia RTCDataChannel es creada, y no se pueden modificar más adelante. [13]

2.4 - Otras tecnologías

2.4.1 - Npm

Npm es un gestor de paquetes para JavaScript. Un gestor de paquetes permite separar el código de una aplicación en distintas partes llamadas paquetes (los cuales tienen distintos archivos en su interior).

El concepto central del uso de paquetes es la creación de pequeños bloques que resuelven un problema concreto, los cuales pueden resolver tareas específicas más complejas formando una unión con otros paquetes. Otra ventaja de usar paquetes y un gestor de paquetes en general es la facilidad de compartir código entre desarrolladores, así como descargar actualizaciones. [16]

2.4.2 - GIT

GIT es un software de control de versiones, siendo el líder en el campo de software libre [17]. En un sistema de control de versiones distribuido, como es el caso de GIT, se tiene la ventaja de poder trabajar sin conexión y sincronizar en algún momento posterior, ya que todas las modificaciones se guardan de manera local. Para la realización de este trabajo hemos usado dos plataformas web basadas en GIT: GitLab y GitHub.

GitLab es un gestor de repositorios web con múltiples funcionalidades adicionales, entre ellas una 'wiki' y un sistema de seguimiento de errores. Desde el inicio de este trabajo se dispuso de un repositorio en GitLab en el que se fueron guardando todos los progresos en el código de manera periódica (cada semana aproximadamente). La principal ventaja que proporciona GitLab respecto a GitHub es que el repositorio en GitLab es privado (funcionalidad de pago en GitHub).

GitHub es otro gestor de repositorios web que, aunque funciona de manera parecida a GitLab (basado en GIT y con funcionalidades adicionales), difiere en un punto básico: todo el código alojado en GitHub es público (salvo que se use la versión de pago). Todo el código desarrollado para el proyecto reTHINK (de manera conjunta entre las empresas pertenecientes al proyecto) está alojado en su repositorio de GitHub, al cuál accedimos de manera constante durante la realización del trabajo.

2.4.3 - Docker

Docker es una plataforma de contenedores de software que nos permite ejecutar un entorno de desarrollo en contenedores separados, siendo la más usada en el mundo [18]. Estos contenedores mantienen una configuración pre-seleccionada, por lo que no hay necesidad de configuración cada vez que arrancamos nuestro entorno.

En este proyecto, usamos docker-compose para ejecutar varios contenedores a la vez, dependiendo el número del entorno usado. En el modo de desarrollo se requieren seis contenedores distintos, para ejecutar los siguientes componentes: catalogue database, registry domain, toolkit (ver capítulo x.x), message node, catalogue broker and proxy. Con un solo comando (docker-compose up) toda la configuración se establece en segundos. En docker-compose, un archivo de extensión .yaml mantiene la configuración

Figure 12: Línea de comandos para el arranque de docker-compose

de todos los contenedores.

El punto positivo principal de el uso de Docker es la independencia de fuentes externas para crear un entorno en el que ejecutar nuestra aplicación: si está instalada una versión estable, no hay necesidad de usar ninguna fuente externa (por ejemplo, la base de datos del catálogo). La consecuencia de esto es que el desarrollo de la aplicación puede continuar mientras otros recursos externos se actualizan o están en mantenimiento.

La desventaja más importante de Docker a la hora de ejecutar nuestra aplicación es la conexión con plataformas externas: las versiones del catálogo y del registro de nuestra aplicación en docker-compose y de la aplicación externa con la que interactuamos deben coincidir. Cualquier actualización en cualquiera de los contenedores afecta al funcionamiento del resto, por lo que todos los contenedores tienen que estar actualizados con la misma versión (o versiones compatibles entre sí).

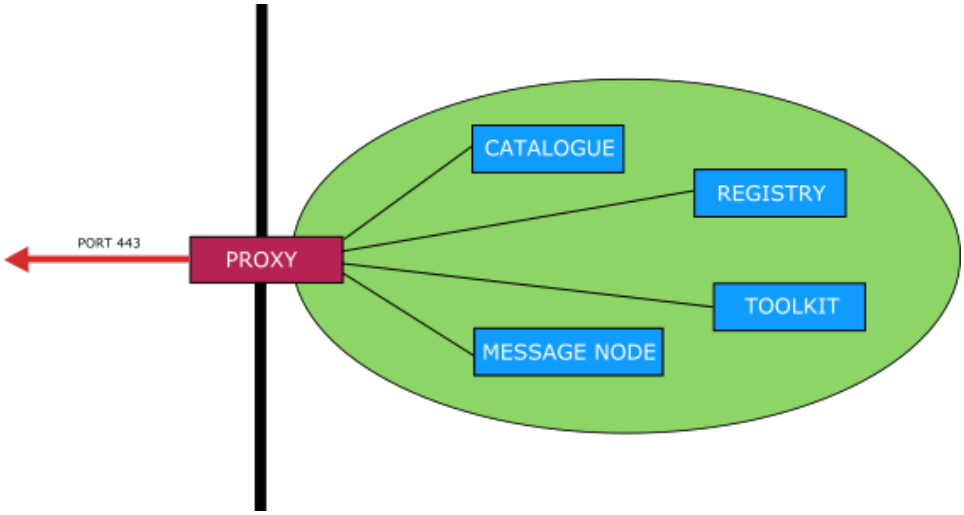


Figure 13: Esquema gráfico: Configuración de los contenedores

Capítulo 3: Arquitectura del desarrollo

En esta sección haremos una descripción del diseño de la aplicación. En primer lugar presentaremos los componentes necesarios según los objetivos descritos al principio de esta memoria. El objetivo principal de la aplicación indica que ésta debe tener un servicio de comunicación con tres funcionalidades principales: motor de búsqueda, establecimiento de conexión con el usuario deseado a través de un chat y de videollamada, así como de establecer una conexión cuando sea otro usuario el que mande la invitación.

Estos requisitos llevan a la elaboración del diagrama presente en la figura número 14), que proporciona una primera aproximación al objetivo encomendado.

A la hora de plantear el diseño, primero se tienen en cuenta las funcionalidades necesarias ya descritas. Después, se elabora un esquema de funcionamiento (descrito en el diagrama presente en la figura número 14). A partir de este esquema se diseña la interfaz gráfica de la aplicación, y después 'se le da vida' a la aplicación, es decir: se implementan los elementos de la arquitectura reTHINK.

Por lo tanto, el énfasis del diseño, apartado descrito en éste capítulo, se

enfoca sobre la interfaz gráfica. Posteriormente se implementan y adaptan a la interfaz los distintos elementos de la arquitectura reTHINK que hacen que la aplicación funcione, proceso descrito de manera detallada en el Capítulo 5. Estos dos elementos condicionan la estructura de archivos elegida, descrito al final de este capítulo.

3.1 - Diseño de la interfaz gráfica

A nivel gráfico, el diseño planteado es el siguiente: La aplicación se basa en 4 escenarios con pantallas de distintos tipos: Una pantalla inicial de bienvenida, una de búsqueda y muestra de resultados, una de chat y una de videollamada. Además, tendremos dos pantallas auxiliares que soportan las funciones de recibir una invitación a chat, y de recibiri una invitación a videollamada. Por lo tanto, haremos uso de 2 hypertys: uno de chat, y otro de videollamada. Pasamos a describir cada uno de los módulos mencionados.

3.1.1 - Pantalla inicial

Es la pantalla mostrada al usuario cuando accede a la aplicación. En ella se deberá mostrar una barra de búsqueda que el usuario acceda a la pantalla de búsqueda y muestra de resultados, además de dar información al usuario al respecto de si la sesión se ha iniciado correctamente o no.

Una pantalla adicional de inicio de sesión para el usuario (en donde introducir nombre y contraseña) no es necesaria, ya que el 'Hyperty Runtime' implementa la gestión de identidades descrita en la sección dedicada a ello.

Por lo tanto, en esta parte de la ejecución se deberá poner en marcha el 'Hyperty Runtime' en segundo plano de manera automática, sin que el usuario tenga que pulsar ningún bóton. Éste pondrá en marcha el gestor de identidades, en el cúal nos pedirá introducir una identidad por cada hyperty utilizado (en este caso, 2), y una vez el proceso haya finalizado con éxito se indicará al usuario que está conectado a reTHINK correctamente.

Éste debe ser un proceso único por cada ejecución: sólo se efectua cada vez que arrancamos de nuevo la aplicación (abriendo una nueva pestaña en el navegador o refrescando la existente).

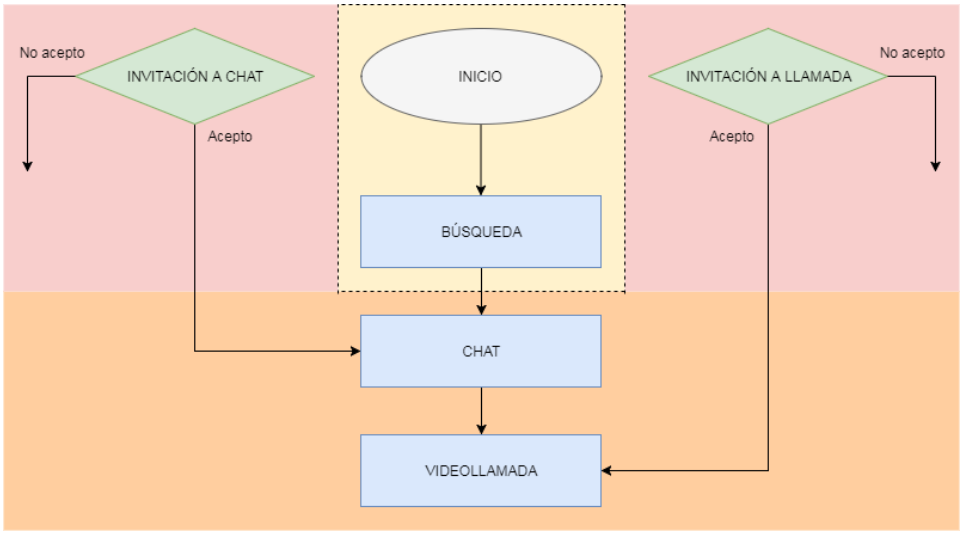


Figure 14: Diagrama de funcionamiento de la aplicación

3.1.2 - Búsqueda y muestra de resultados

A esta pantalla se accede desde la pantalla inicial realizando una búsqueda. En esta pantalla se deben mostrar los resultados que ha dado la búsqueda, así como realizar una nueva búsqueda tantas veces como sea necesario, refrescando con nueva información la pantalla que muestra los resultados.

Debido a la complejidad de la información del usuario, la decisión de diseño es desplegar una lista de resultados en la que sólo se muestran los nombres de estos resultados (en el caso de nuestra aplicación, los nombres de los usuarios). Para conocer más detalles de un determinado usuario, así como para establecer una conexión con el, se deberá desplegar una nueva ventana que ofrezca estas posibilidades. Este modo de funcionamiento se puede ver de manera esquemática en la figura 15.

Es un requisito que en la información del usuario se ofrezcan las opciones de conexión disponibles Es decir, se deben mostrar al usuario de una manera fácil de entender (sin conocer los conceptos reTHINK) qué hypertys están activos, y por tanto que servicios están disponibles para un determinado usuario, así como la posibilidad de mandar una invitación o conectar directamente con ese servicio.

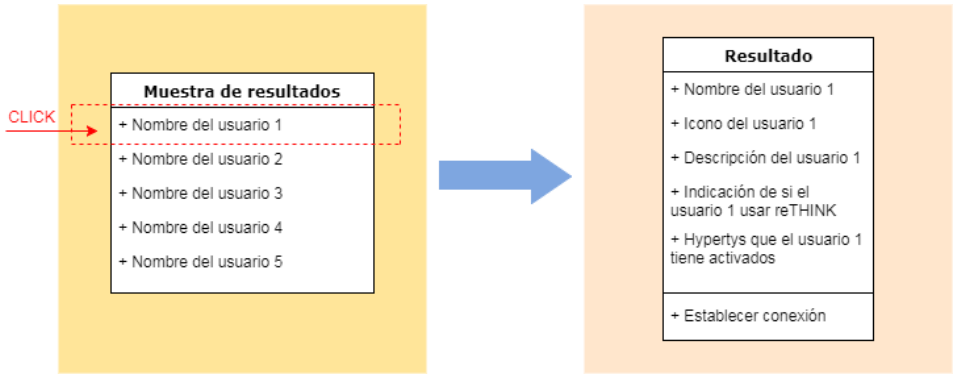


Figure 15: Esquema de muestra de resultados

El motor de búsqueda implementado deberá ser 'Global Discovery', desarrollado para reTHINK.

3.1.3 - Chat

La ventana de chat se abre cuando, o bien se invita a un usuario encontrado en una búsqueda, o bien se recibe la invitación de un usuario (proceso que se describe dos secciones más adelante). Ésta ventana debe desplegar las funcionalidades más básicas de una aplicación chat: Cuadro de texto en el cual se muestra la conversación, cuadro de entrada de texto en el cual escribimos el mensaje a enviar, y botón de envío.

Además, como funcionalidades específicas de nuestra aplicación se deberá mostrar el nombre del usuario con el que estamos hablando y la posibilidad de establecer una videollamada con ese usuario.

Esta pantalla deberá desplegar el hyperty de chat, el cual ya está pensado para poder establecer varios chats con distintos usuarios al mismo tiempo, sin la necesidad de realizar grandes modificaciones en el código.

Para crear una aplicación eficiente, se deberá liberar el recurso utilizado cuando la conversación de chat llegue a su fin.

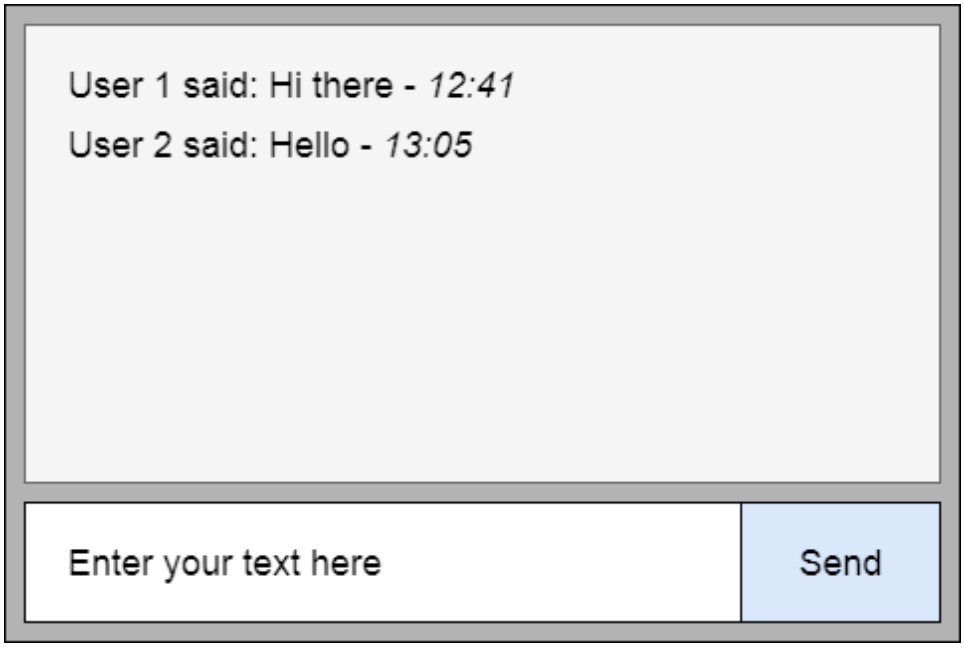


Figure 16: Modelo de ventana de chat que cumple las especificaciones

3.1.4 - Videollamada

De manera análoga a la ventana de chat, la ventana de videollamada se abre cuando, o bien se invita a un usuario a través de la ventana de chat, o bien se recibe la invitación de un usuario. Ésta ventana deberá implementar los elementos básicos de una videollamada (cámara propia y del otro usuario y el sonido del otro usuario), así como las funcionalidades del chat descritas anteriormente.

Además de los elementos más básicos de una videollamada, la aplicación deberá poder controlar el volumen del sonido, apagar la cámara y el micrófono propios y finalizar la llamada. Como último requisito, las ventanas de videollamada y chat se deberán mostrar a la vez, sin tener que cambiar de una a otra, como se puede apreciar en la figura 17.

La pantalla de videollamada deberá desplegar el hyperty de videollamada DTWebRTC creado para reTHINK. Debido a que éste hyperty no soporta varias conversaciones simultáneas con distintos usuarios, se deberá desarrollar un hyperty que sí lo permita. Además de este hyperty, se mantendrá en

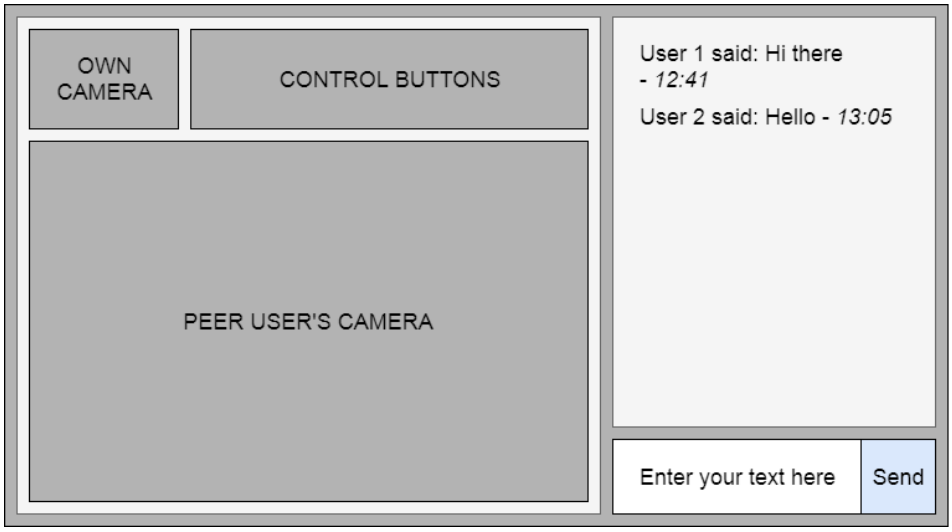


Figure 17: Modelo de ventana de videollamada que cumple las especificaciones

funcionamiento el hyperty de chat que teníamos con anterioridad si nuestro usuario manda la invitación, o se establecerá un nuevo hyperty de chat si nosotros recibimos la invitación.

Como en el caso del chat, para crear una aplicación eficiente, se deberá liberar el recurso utilizado cuando la conversación de chat llegue a su fin. En este caso, si cerramos la videollamada volveríamos a la ventana de chat (figura 16).

3.1.5 - Ventanas de invitación

En el caso de utilización de que, una vez la pantalla de inicio haya puesto en marcha el 'Hyperty Runtime' con éxito, recibamos una invitación de conexión por parte de otro usuario, deberemos gestionar la invitación y abrir la pantalla con el recurso requerido en caso de que la invitación sea aceptada.

Aunque para recibir una invitación el proceso realizado en la Pantalla de Inicio debe ser finalizado con éxito, la aplicación debe gestionar las invitaciones recibidas de manera correcta en cualquier momento de la ejecucón (cuando tengamos otro chat abierto o estemos realizando una búsqueda, por

Figure 18: Esquema de muestra de resultados

ejemplo).

En el caso de la invitación a chat, se deberá abrir una ventana emergente que contenga los campos de información más relevantes del usuario que la envía, y deberá abrir la nueva ventana de chat en segundo plano. La ventana emergente tendrá un botón de cerrar, en el que si hacemos click podremos regresar al punto donde estábamos de nuestra aplicación.

Para el caso de la invitación a una videollamada, también se abrirá una ventana emergente que contenga información del usuario que llama, pero en este caso la videollamada no se establecerá automáticamente: la ventana emergente tendrá un botón de aceptar la llamada y otro de declinarla. Al pulsar cualquiera de los dos botones la ventana emergente se cerrará.

3.1.6 - Sistema de pestañas

Debido a que los requisitos de la aplicación establecen que debe ser capaz de mantener varias conversaciones a la vez (lo cual requiere más de una

ventana), y teniendo en cuenta que en el momento de realización de este trabajo no era posible desplegar un mismo 'Hyperty Runtime' en distintas pestañas del navegador, para mantener las distintas pantallas abiertas simultáneamente es necesario crear un sistema de pestañas dentro de la pestaña del navegador (como indica la figura 18).

En la figura 14 se puede ver que los distintos módulos están encuadrados en áreas pintadas de determinados colores. En el área pintada de amarillo se encuadran la pantalla inicial y la de búsqueda, las cuales sólo se abren una vez. La pantalla inicial se muestra únicamente al principio, y desaparece en cuanto se abre alguna pestaña. La ventana de búsqueda se abre como una nueva pestaña que no tiene botón de cerrar, por lo que sigue abierta hasta el final de la ejecución.

Las ventanas encuadradas en el área pintada de naranja son la de chat y la de videollamada, y se pueden abrir tantas ventanas como sean necesarias (siempre que sean chats y videollamadas con distintos usuarios, ya que no se pueden abrir dos pestañas con la misma conversación).

Y por último, las ventanas encuadradas en el área pintada de color rojo son las correspondientes a las invitaciones, que se abren en modo de ventana emergente (y no de pestaña) y se cierran cuando han cumplido su función.

En una posterior versión de la aplicación el objetivo sería desplegar estas distintas ventanas usando las pestañas del navegador, opción no abordada en este trabajo debido a las razones anteriormente expuestas.

3.2 - Uso de elementos reTHINK

A la hora de efectuar el diseño de la aplicación, se tuvieron en cuenta los elementos reTHINK básicos a utilizar de manera general, haciendo un listado de ellos y sus requerimientos. A nivel más específico, el proceso de cómo estos elementos y sus funciones y variables encuadran con la interfaz gráfica se da durante el desarrollo (Capítulo 5). En cualquier caso, a continuación se enumeran éstos elementos.

En la aplicación desarrollamos un elemento de comunicación de videoconferencia y de chat, por tanto haremos uso de dos hypertys, que cubran esas dos áreas respectivamente. Los dos hypertys están disponibles en el

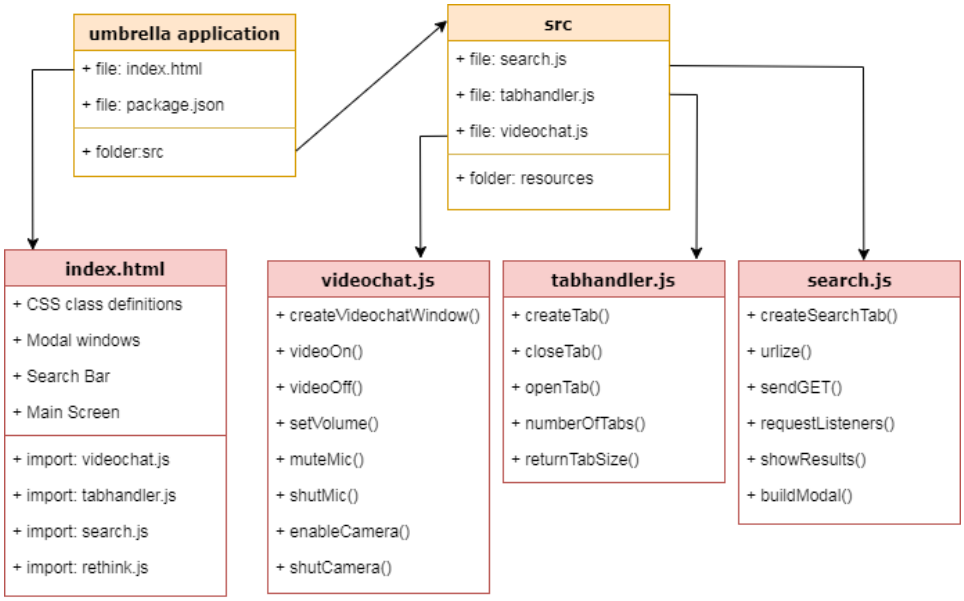


Figure 19: Estructura de archivos de la aplicación

repositorio oficial de reTHINK en GitHub. Gracias a Docker, no necesitamos instalar estos hypertys en nuestro código, pues quedan instalados en contenedores separados. En la última fase del desarrollo se modifica el hyperty de videollamada, cuyo diseño de la estructura de código se puede ver al final del Capítulo 5.

También implementamos un motor de búsqueda el cual no requiere del diseño de una estructura de código para él debido a que para su funcionamiento no es necesaria ninguna instalación en nuestro código. Tampoco la rquiere el mecanismo de gestión de identidades, ya que es procesado automáticamente por el Hyperty Runtime, cuya instalación en nuestra aplicación explicaremos más adelante.

3.3 - Estructura de los archivos

Teniendo en cuenta los distintos puntos expuestos en las dos secciones anteriores, se diseña la estructura de archivos que se puede ver en la figura número 19.

En la carpeta 'umbrella application', que contiene todos los archivos de la aplicación necesarios, separamos por un lado el archivo HTML (index.html) que arranca la aplicación en un primer momento y define los elementos gráficos, y los archivos Javascript (search.js, tabhandler.js y videochat.js) que definen cómo interactúan esos elementos entre sí.

Además, de manera auxiliar se encuentra el archivo package.json, que incluye instrucciones sobre la ejecución del Hyperty Runtime, y la carpeta resources, que incluye todos los iconos utilizados en nuestra aplicación.

Debido al funcionamiento de nuestra apliación definido en la figura número 14, muchos de los elementos gráficos de nuestra aplicación deberán ser creados en Javascript de forma dinámica (y no en HTML de forma estática al principio de la ejecución). Por ejemplo, durante el uso de nuestra aplicación puede que no usemos la función de videollamada, por lo tanto los elementos gráficos utilizados durante la videollamada deberán ser creados de forma dinámica, únicamente si son necesarios.

Aunque en el momento de diseño de la arquitectura de la aplicación no quedaron definidas todas las funciones incluidas en la figura número 19 (lo cual se hizo durante el desarrollo de la aplicación), la estructura de archivos, sus funciones y sus relaciones entre sí fueron diseñados desde un primer momento.

Capítulo 4: Desarrollo previo

Los objetivos de esta fase inicial del desarrollo son conocer la viabilidad de crear un hyperty que pueda cumplir simultáneamente los roles de 'Reporter' y 'Observer' descritos en la sección de estado del arte. Para ello, crearemos un nuevo hyperty llamado 'Slider', que aunque no será implementado en la aplicación de la que es objeto este Trabajo de Fin de Grado, las conclusiones de la viabilidad antes mencionada las emplearemos en la aplicación, además de utilizar el mecanismo de sincronización desarrollado.

También ha servido para conocer y entender los principios de reTHINK de manera práctica, relevante en la elaboración del capítulo de estado del arte. Por último, usaremos la herramienta de desarrollo de hypertys 'Hyperty Toolkit' creada por reTHINK para comprobar su utilidad a la hora de desarrollar la aplicación.

4.1 - Hyperty Toolkit

En primer lugar, instalamos la herramienta 'Hyperty Toolkit' y nos familiarizamos con ella.

'Hyperty Toolkit' es un espacio de trabajo creado por el proyecto reTHINK en el cual los desarrolladores puede probar los hypertys que ellos mismos han creado. 'Hyperty Toolkit' está escrito en JavaScript, y proporciona todos los elementos necesarios para establecer un entorno reTHINK que incluye los elementos que hemos definido previamente (Hyperty Runtime, una base de datos con distintos hypertys ya desarrollados, un catálogo, un registro, etcétera).

Los hyperties ya desarrollados que podemos usar en la 'Hyperty Toolkit' son entre otros DTWebRTC (hyperty de videollamada que usaremos en nuestra aplicación), Hello World (el cual usaremos y modificaremos en esta sección), Bracelet Sensor o Room Service.

Para instalar la 'Hyperty Toolkit', debemos usar el repositorio oficial de reTHINK en GitHub [19] en el cual se encuentran todos los archivos necesarios. Primero clonamos el repositorio de la 'Hyperty Toolkit' a nuestro disco local, así como el repositorio donde se incluyen los hypertys ya creados en una carpeta paralela. Después, tenemos que instalar los módulos de la 'Hyperty Toolkit' a través del gestor de paquetes 'npm' (descrito en la

Figure 20: Clonación de repositorios para la instalación de la 'Hyperty Toolkit'

sección 'Estado del Arte').

Una vez instalada la 'Hyperty Toolkit', ponemos en marcha el servidor HTTP local y el catálogo local con el comando 'npm run start:browser' (figura 21), seleccionando el directorio donde se encuentra la base de datos de hypertys. 'Hyperty Toolkit' desplegará todos los hypertys que se encuentren en esa base de datos. Y cuando se ha completado el proceso de carga de hypertys, ya podemos acceder a la hyperty toolkit desde el navegador a traveés del host local, puerto 443 (https://localhost:443). Como último requisito, para que la 'Hyperty Toolkit' funcione, se deben aceptar los certificados de la misma, así como de su catálogo.

La primera ventana que ofrece la 'Hyperty Toolkit' es la mostrada en la figura 22, en la que podemos seleccionar de un listado el hyperty que queremos usar. Antes de que aparezca el listado desplegable, la 'Hyperty Toolkit' ya está desplegado el 'Hyperty Runtime' en segundo plano. Aunque el 'Hyperty Runtime' permite ejecutar más de un hyperty como veremos más adelante, en la 'Hyperty Toolkit' sólo podemos usar un hyperty al mismo tiempo.

Hyperty Hello World

El primer hyperty que usaremos en la 'Hyperty Toolkit' es el llamado 'Hello World'. Este hyperty enseña las funcionalidades más básicas de reTHINK, existiendo un hyperty con el rol 'Reporter' y otro con el rol 'Ob-

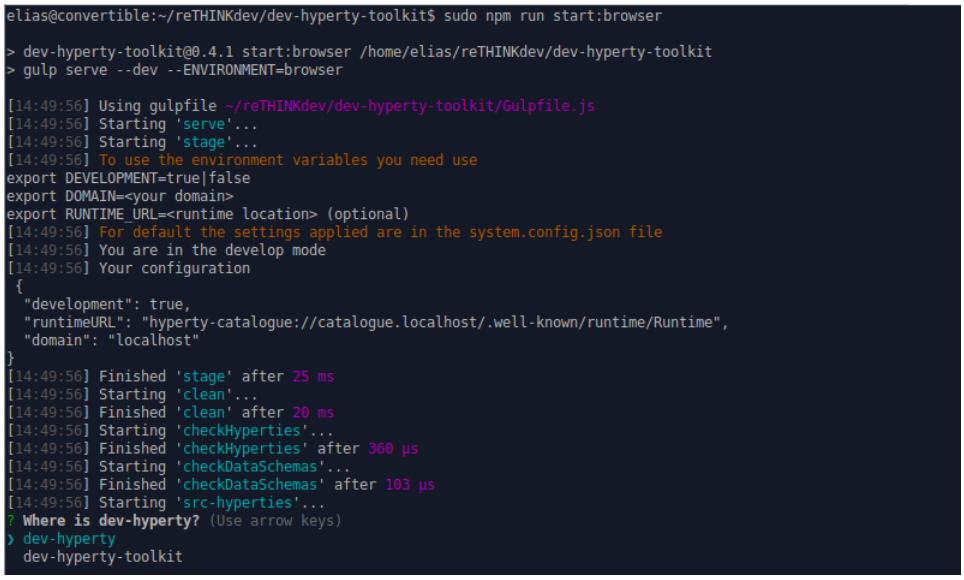


Figure 21: Arranque de la 'Hyperty Toolkit'

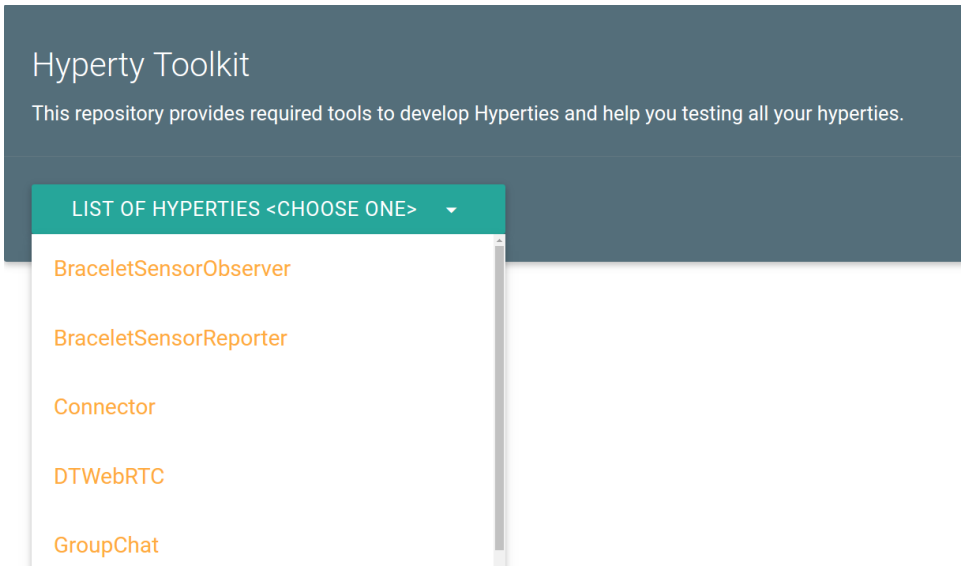


Figure 22: Selección de hypertys a ejecutar en la 'Hyperty Toolkit'

server'. El hyperty con el rol Reporter manda un mensaje de bienvenida al 'Observer', y después edita el objeto de datos previamente compartido (enviando un mensaje de despedida).

El esquema de comunicación con el hyperty Hello World es el modelo de comunicación descentralizada explicado anteriormente (ver figura 7). Para este caso, como podemos ver en la figura 23, cada hyperty tiene un archivo JavaScript que define su operaciones (siendo distinto el Reporter del Observer).

Para el caso del Hello World Reporter, éste archivo se llama HelloWorldReporter.hy.js. Contiene un constructor, la creacón del objeto de datos, una función hello (crea el sincronizador y manda la invitación al Observer) y una función bye (que modifica el objeto de datos enviando el mensaje de despedida).

En el caso del Hello World Observer, el archivo que implementa las funcionalidades del hyperty se llama HelloWorlObserver.hy.js. Contiene un constructor y una función onNotification, la cual recibe la invitación del Reporter y actualiza el objeto de datos cada vez que éste es modificado.

Además, es necesario definir el objeto de datos a utilizar. El archivo HelloWorldDataSchema define la estructura a seguir del objeto de datos, y el archivo hello.js le da valores, siendo este último el objeto de datos que manejan los hypertys Reporter y Observer. Por último, dos funciones construyen en JavaScript la interfaz necesaria para la operación del hyperty Hello World: helloReporter.js y helloObserver.js.

Es relevante señalar que el hyperty 'Hello Wolrd' carece de sistema de búsqueda de usuarios, lo cual limita las posibilidades de aplicación. En un caso práctico de funcionamiento, en las ventanas del navegador abiertas para el Reporter y para el Observer se muestran las respectivas URLs de los hypertys. Para establecer una conexión, debemos copiar la URL del Observer y proporcionársela al Reporter. Una vez realizado este paso ya podemos mandar el mensaje de bienvenida, como

4.2 - Slider Hyperty

Una vez desplegada la 'Hyperty Toolkit' y el hyperty Hello World, procedemos a crear un nuevo hyperty llamado 'Slider Hyperty' que implementa la

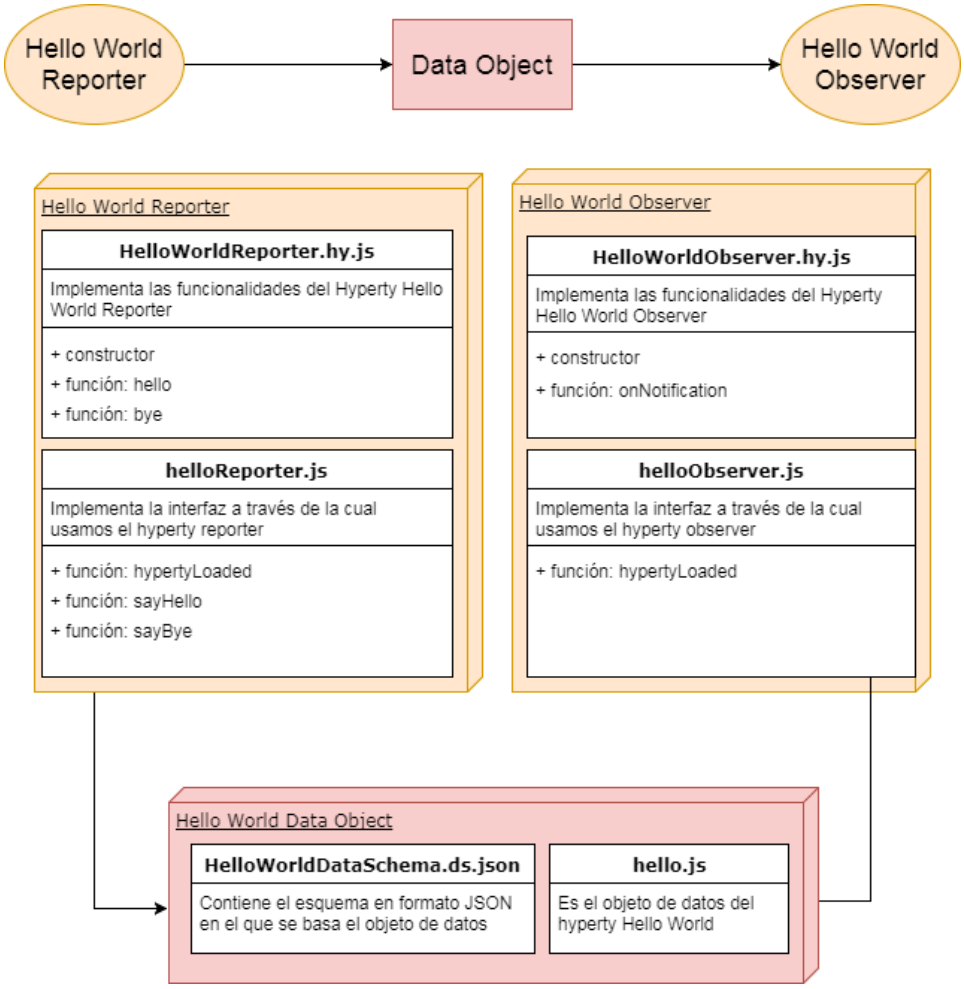


Figure 23: Esquema de funcionamiento del hyperty Hello World y los archivos que lo componen

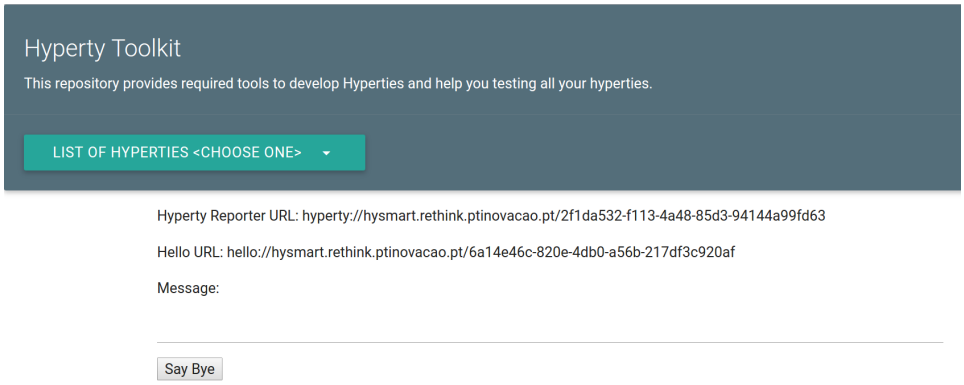


Figure 24: Captura de pantalla del hyperty Hello World Reporter

función de un sensor. Sobre la base del hyperty Hello World, éste hyperty despliega un deslizador (similar a una barra de control de volumen) en los dos usuarios, detectando cualquier modificación en el deslizador y actualizando automáticamente el nuevo valor en el deslizador del otro usuario.

Para la demostración operativa de este nuevo hyperty usaremos la 'Hyperty Toolkit'. En el desarrollo de este hyperty nos basaremos en la estructura de archivos del hyperty Hello World vista en la sección anterior, pero incluyendo una modificación: debido a que los cambios en el deslizador pueden ocurrir en cualquiera de los dos usuarios, este hyperty será bi-direccional (a diferencia del hyperty 'Hello World', con roles Reporter-Observer diferenciados).

Sincronización

Para conseguir la conexión bi-direccional, debemos crear dos sincronizadores: El primero lo crea el slider que inicia la conexión, el cual le manda la instancia del sincronizador al otro usuario, y este se subscribe. Al recibir la instancia del sincronizador, este usuario modifica su interfaz a través de la función trigger para mostrár el deslizador. Una vez que este proceso ha terminado, este último usuario crea otro sincronizador y se lo envía al primer usuario, que se subscribe. De esta manera, tenemos conexión en los dos sentidos. Este proceso se puede ver de manera gráfica en la figura 26.

Código fuente del hyperty Slider

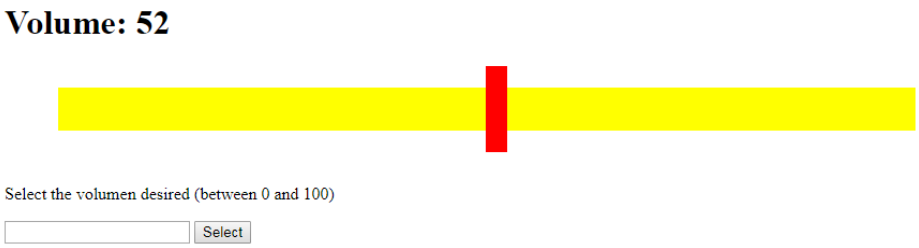


Figure 25: Deslizador del hyperty Slider

Figure 26: Mecanismo de sincronización del hyperty Slider

En cuanto a la estructura de archivos de código de este hyperty, la primera consideración a realizar será la existencia de un solo hyperty: si bien en el hyperty 'Hello World' existían dos grupos de archivos (Reporter y Observer) con estructura similar (ver figura 23), en el hyperty 'Slider' no existe esa necesidad: existirá solo un grupo de archivos que implemente las funciones de Reporter y Observer, y los dos usuarios usarán el mismo grupo de archivos: slider.hy.js y slider.js.

El archivo slider.js.hy implementa las funcionalidades necesarias para el hyperty Slider, y se vale para ello de un constructor y 3 funciones distintas, como se puede ver en la figura 27.

En primer lugar, el constructor crea los elementos necesarios que se utilizarán a lo largo de la conexión, entre ellos el sincronizador (crea el sincronizador de manera local).

La primera función, onNotification, se pone en marcha nada más crear el sincronizador, y permanece a la escucha. En cuanto recibe una instancia del sincronizador de otro usuario, se subscribe a ese sincronizador y envía los mensajes pertinentes a la interfaz de usuario usando la función trigger. Además, valiéndose de la función 'connect', manda la instancia de su propio sincronizador al otro usuario. Una vez terminado este proceso, permanece a la escucha de cambios en el objeto, los cuales modificarían la posición del deslizador.

La función connect, como ya se ha mencionado, crea y manda una instancia del sincronizador propio al otro usuario con el que queremos establecer la conexión. Esta función puede ser llamada antes de establecer la conexión, siendo la que desencadena el mecanismo de sincronización mostrado en la figura 26, o desde la función onNotification como se explica en el párrafo anterior.

Por último, sendElement recibe de la interfaz gráfica un nuevo valor del deslizador, y actualiza el valor del objeto de datos con este nuevo valor. Debido al mecanismo de sincronización, el otro usuario recibirá automáticamente la modificación del valor y actualizará el deslizador de su interfaz gráfica.

Objeto de datos

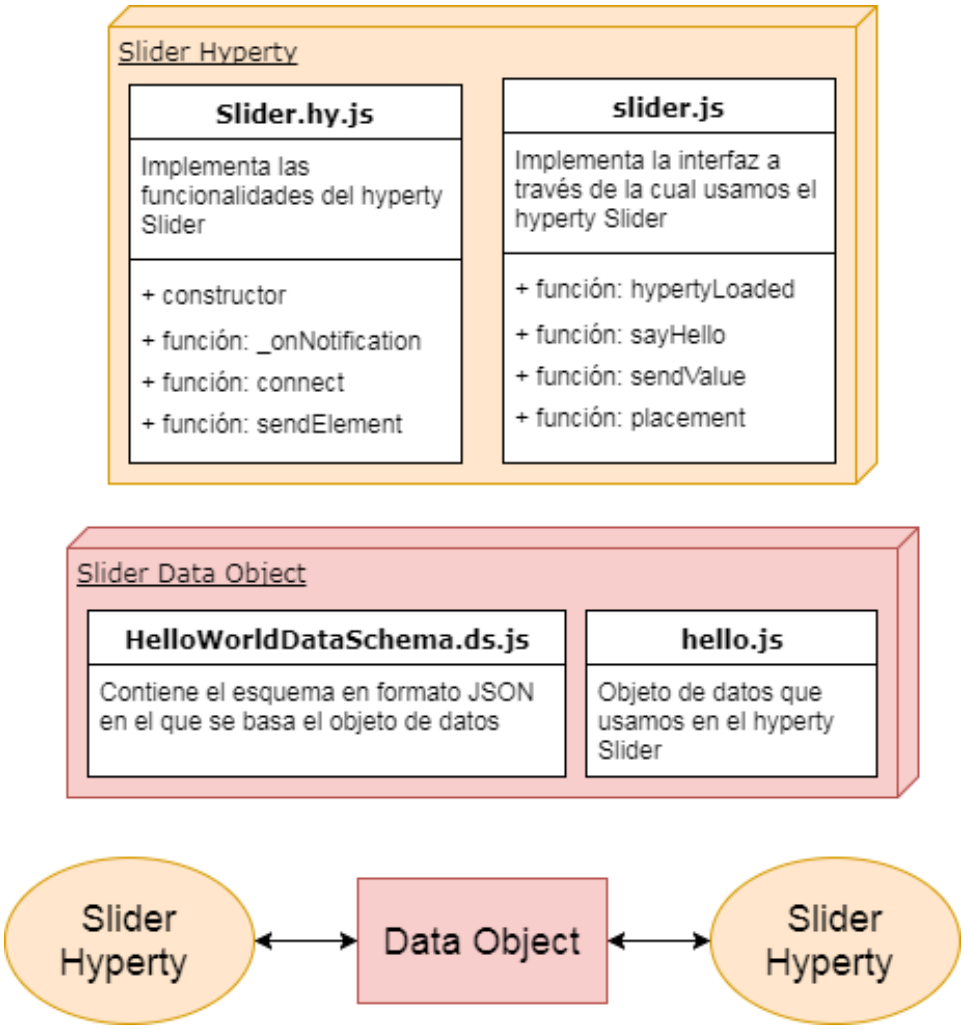


Figure 27: Esquema de funcionamiento del hyperty Slider y los archivos que lo componen

Respecto al objeto de datos, aprovecharemos la misma estructura del usado en el hyperty 'Hello World' sin modficación alguna en el formato, con la única salvedad de que en el hyperty slider el valor será numérico, a diferencia del hyperty 'Hello World' en el que el valor era una palabra. Por ello podemos comprobar que estos dos archivos quedan intactos comparando las figuras 23 y 27, manteniendo incluso los nombres. El valor del objeto hello.js por defecto es 0, iniciándose los dos deslizadores con este nivel, y pudiendo tomar valores entre 0 y 100.

Interfaz gráfica

Por último, necesitaremos crear un archivo JavaScript que implemente gráficamente el deslizador. Éste archivo se ha creado de cero sin el uso de plantilla alguna, y se puede ver en la figura 25. El valor de la posición del deslizador se puede introducir de dos maneras: bien pinchando con el ratón en cualquier parte (de color amarillo) del deslizador, o bien introduciendo un número en el cuadro de texto.

El archivo slider.js implementa la interfaz de usuario del deslizador, así como las funciones necesarias para darle interactividad. Estas funciones son 4: hypertyLoaded, sayHello, sendValue y placement.

hypertyLoaded crea una instancia del hyperty con la que trabajeremos a lo largo de este archivo, y prepara dos 'listeners' (escuchadores) que reciben la señal de las funciones trigger que se pueden observar en la figura 26. El 'listener' invitation transmite la información al usuario de que se ha recibido una invitación cerrando la ventana inicial, y el 'listener' hello despliega el deslizador una vez que la sincronización se ha producido. A partir de este momento los dos deslizadores están conectados y listos para interactuar.

La función sayHello se emplea para iniciar una conexión con otro usuario del que conocemos la URL de su hyperty, ejecuta la función 'connect' descrita anteriormente y por tanto desencadena la conexión. A nivel de interfaz, sayHello despliega el deslizador.

sendValue se ejecuta cuando se recibe un cambio de posición en el deslizador efectuado por el propio usuario (moviendo el deslizador o introduciendo un valor en la caja de texto), y llama a la función sendElement, que efectúa la modificación en el objeto de datos.

placement es una función auxiliar con propósitos exclusivamente relacionados con la interfaz gráfica, y se encarga de actualizar visualmente la posición del deslizador cuando se produce algún cambio, bien sea por el propio usuario o bien sea por el otro usuario.

Capítulo 5: Desarrollo y validación de la aplicación

Esta es la sección más extensa de este Trabajo de Fin de Grado, y en ella pasaremos a describir detalladamente cuales han sido los pasos dados en el desarrollo de la aplicación. Éstos pasos se mencionan en la sección 'Fases de Realización' (1.3), y son los siguientes: Creación de la aplicación, Creación del servicio de comunicación para la aplicación, y desarrollo de un nuevo hyperty de videollamada múltiple para la aplicación.

En la primera fase se creará la aplicación, implementando el 'Hyperty Runtime' y todas las funcionalidades reTHINK asociadas, como el catálogo o el registro. También deberá tener una función de búsqueda de usuarios haciendo uso del sistema 'Global Discovery'.

La segunda fase de desarrollo tiene por objetivo implementar los ya desarrollados hypertys de videollamada y chat para hacer funcional la aplicación que ya está creada.

En la tercera y última fase de desarrollo el objetivo es crear un nuevo hyperty que soporte varias videollamadas e implementarlo en la aplicación, para así cumplir con los objetivos establecidos al inicio de este Trabajo de Fin de Grado.

5.1 - Creación de la aplicación

El objetivo general es crear una aplicación web que use el Hyperty Runtime (sistema en tiempo de ejecución para hypertys) y sea capaz de desplegar un hyperty y sus funcionalidades asociadas (catálogo, registro)

5.1.1 - Configuración del entorno de trabajo

La primera tarea a la hora de crear la aplicación sera configurar un entorno de desarrollo que permita ejecutar los distintos elementos reTHINK (catálogo, registro, etcétera) necesarios para nuestra aplicación. Para ello utilizaremos la aplicación Docker, detallada en la sección de Estado del Arte.

En el repositorio oficial de reTHINK en GitHub se encuentran todos los archivos necesarios para la configuración de Docker, representados en

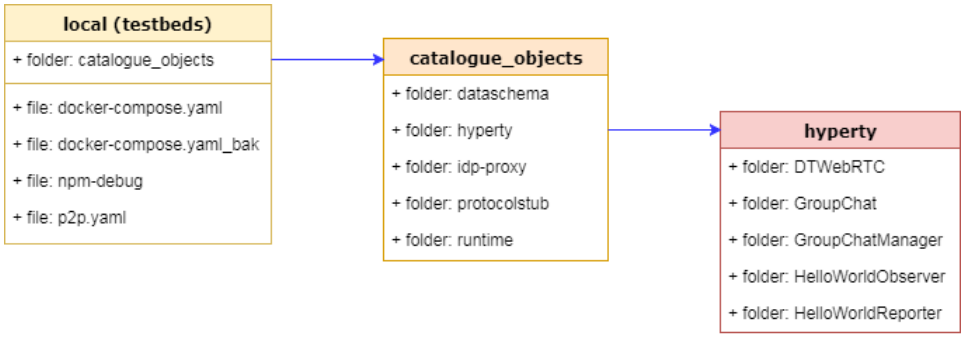


Figure 28: Estructura de archivos de la configuración Docker

la figura número 28. Debido a que estos archivos ya están desarrollados y listos para utilizar con los hypertys existentes, para desarrollar nuestra aplicación lo único que tenemos que hacer es modificar el archivo 'docker-compose.yaml', y si nuestra aplicación utiliza un hyperty desarrollado desde cero (como es el caso de la sección 5.3) deberemos guardarlo en la carpeta 'hyperty'.

El archivo 'docker-compose.yaml' define los distintos contenedores necesarios para ejecutar nuestra aplicación. Los nombres de los contenedores son los siguientes: nomatrix, dev-registry-domain, catalogue-broker, catalogue-database, toolkit y proxy.

Siempre que utilicemos los hypertys ya existentes (y sus archivos relacionados, por ejemplo los incluidos en las carpetas protocolstub o dataschema), la única configuración necesaria es especificar la ruta (en el contenedor 'toolkit') del lugar donde están guardados los archivos Javascript de nuestra aplicación.

5.1.2 - Despliegue del Hyperty Runtime

En nuestro código, la primera funcionalidad a ejecutar será el despliegue del Hyperty Runtime. A nivel de interfaz de usuario, ésta abrirá varias pantallas (tantas como el número de hypertys a desplegar) para seleccionar las identidades de usuario a cada uno de los hypertys.

Para efectuar el despliegue, primero debemos importar el archivo 're-

```
rethink.default.install(config).then(function(result) {
    runtimeLoader = result;
    console.info('Runtime Installed in production mode:', result);

}).then(function(hyperties) {
    loadHyperties();
}).catch(function(reason) {
    console.error(reason);
});
```

Figure 29: Instalación de la Hyperty Runtime

think.js'. Éste contiene toda la información necesaria para desplegar el Hyperty Runtime, y a partir de ahí podremos instalarla invocando a la función 'install' (ver figura número 29).

Una vez instalada, el siguiente paso será desplegar los hypertys sobre ella, lo cual se detalla en secciones posteriores.

5.1.3 - Sistema de pestañas

Para cumplir las especificaciones de la aplicación, es necesario desarrollar un sistema de pestañas que permita la comunicación con distintos usuarios al mismo tiempo, debido a que el Hyperty Runtime no estaba disponible para ser desplegada en distintas pestañas del navegador en el momento de realización de este proyecto.

Las funciones que este sistema de pestañas debe cubrir son las siguientes:

- **Crear pestaña:** Cuando la aplicación lo requiere, una nueva pestaña con su propio contenedor deberá ser abierta. El tamaño de las pestañas en la barra superior se deberá ajustar según el número de pestañas abiertas, sin descuadrar el formato de la aplicación. Hay varios casos que llevan a abrir una nueva pestaña: Cuando el usuario efectúa una búsqueda, cuando se abre una nueva conversación chat con un usuario, cuando se recibe un mensaje de chat desde otro usuario, o cuando se acepta una videollamada por parte de otro usuario.
- **Navegar entre pestañas:** El sistema debe permitir abrir cualquier pestaña en cualquier momento, presentando en pantalla el proceso asociado a la pestaña seleccionada mientras el resto de los procesos se ejecutan en un segundo plano.

- **Cerrar pestaña:** En cada pestaña (salvo en la de búsqueda) debe existir un botón de cierre con el cuál se cierra la pestaña y su contenido, y además detiene el proceso que se ejecutaba en esa pestaña (por ejemplo, una videollamada), liberando los recursos utilizados.

Para desarrollar este sistema, primero se creó un archivo HTML en el cual se definieron los distintos módulos HTML y CSS, y un archivo Javascript con 3 funciones principales y 2 secundarias, como se ve en la figura número 30. Pasamos a describirlo en detalle:

Index.html

El archivo index.html contiene en primer lugar las definiciones de clase en lenguaje CSS necesarias para darle formato a los distintos elementos de la web. Estos tipos de elementos o clases son 3: tab, content y close.

Tab es la barra superior que contiene las distintas pestañas abiertas en nuestra aplicación, definidas justo debajo como tab div. En esta parte del código quedan definidas las características por defecto de las pestañas, como por ejemplo el tamaño y el color de la pestaña, la fuente o el borde. Éstas caracrerísticas pueden cambiar si se posa el cursor sobre ellas (div:hover), o si se hace click en ellas (div:focus,.active).

Content es el bloque que muestra la pantalla asociada a una determinada pestaña. Por ejemplo, si abrimos una nueva pestaña de chat, el elemento content mostraría la conversación y el cuadro de entrada de texto. En la definición de las características por defecto de la clase content se incluyen el tamaño y el color de fondo, entre otros.

Close es el botón representado con una x que cierra la pestaña, y está incluido dentro de cada 'tab div'. Al igual que con los elementos del tipo tab, los elementos del tipo close cambian sus características si el cursor interactúa con ellos.

En la parte final del archivo index.html se define el bloque 'general div', el cual incluye la barra superior que contiene a las pestañas (allTabs, elemento del tipo tab) y el bloque de contenidos (allContent, elemento del tipo content). Como se puede ver en la figura número 30, estos dos elementos son listas HTML no numeradas (del tipo ul). Cada vez que se abre una nueva pestaña, se añade un nuevo elemento a la lista allTabs y un nuevo elemento

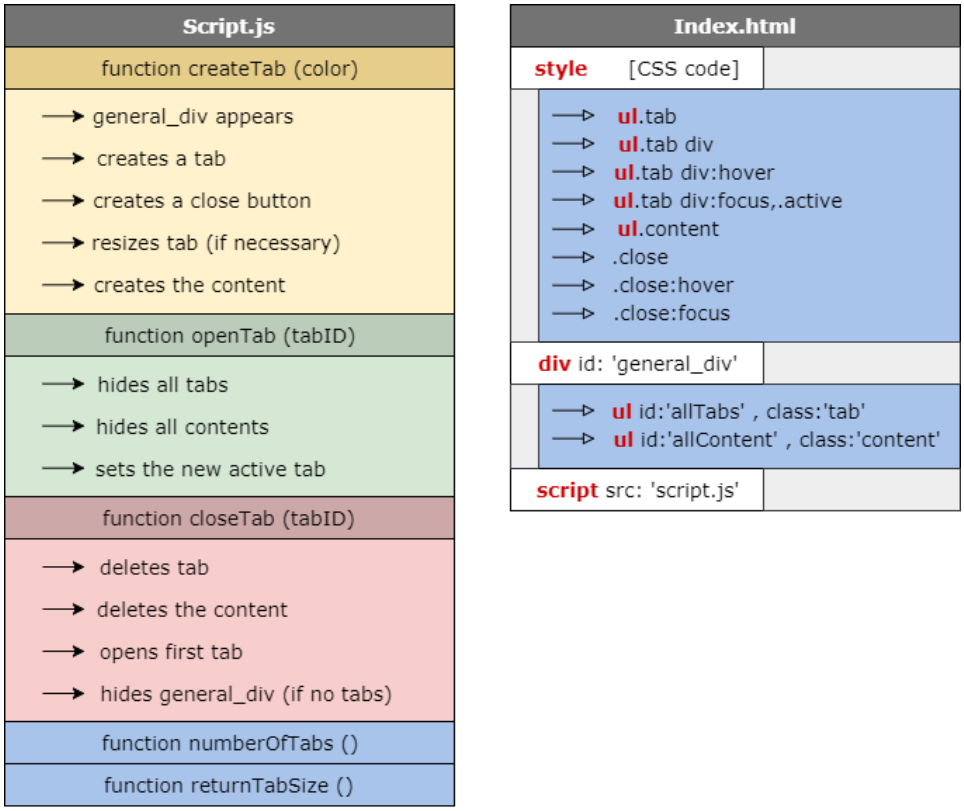


Figure 30: Estructura de código del sistema de pestañas

Figure 31: Ejemplo de sistema de pestañas por colores

a la lista allContent.

Tomando como ejemplo la figura número 31, el recuadro azul representa allTabs, y el recuadro verde un elemento de la lista allTabs (en la práctica, una pestaña). El recuadro amarillo representa un elemento de la lista allContent. En este caso, el color del contenido es rojo, ya que hemos seleccionado la pestaña 'lightcoral'. También se puede apreciar que la pestaña 'lightcoral' es la seleccionada o activa ya que el color de fondo es más oscuro que el resto de pestañas.

Script.js

Mientras que el archivo index.html define los elementos gráficos de nuestro sistema de pestañas, el archivo Javascript script.js define cómo esos elementos interactúan entre sí. La estructura de este archivo se puede ver en la figura número 30.

La primera función, createTab, abre una nueva pestaña y crea su contenido. Para el ejemplo con pestañas de colores de la figura número 31, createTab recibe como parámetro el color del contenido asociado a la pestaña (por ejemplo 'lightcoral' en la figura).

El primer paso que realiza la función createTab es mostrar el bloque 'general div' en caso de que aún no haya ninguna pestaña abierta. Después, crea un elemento de la clase 'tab div' cuyo texto será el color recibido como parámetro y lo añade la lista 'allTabs', así como el botón de cerrar. En el caso de nuestra aplicación, el texto de la pestaña será el tipo de hyperty

(chat, videollamada...).

A continuación comprueba que el número de pestañas sea igual o inferior a 4. Si es mayor, ajusta el tamaño de todas las pestañas para que el formato no quede descuadrado. Para ello utiliza las funciones auxiliares numberOfTabs y returnTabSize, que en función del número de pestañas abiertas calcula el tamaño que éstas deben tener.

Por último crea el bloque de contenido asociado a la pestaña, añadiéndolo como un nuevo elemento en la lista 'allContent'. Para el ejemplo de las pestañas de colores, el bloque de contenido será un recuadro del color recibido como parámetro al inicio de la función.

La segunda función, openTab, es la que nos permite seleccionar la pestaña de la que queramos mostrar su contenido, recibiendo como parámetro el identificador de la pestaña. Para llamar a esta función basta con hacer click sobre una pestaña (cualquiera de los elementos incluidos en la lsita 'allTabs'). Esta funcionalidad queda definida cuando se crea la pestaña con la función createTab.

En primer lugar, desactiva todas las pestañas (elementos de la lista 'allTabs') que puedan estar marcadas como activas, y después realiza lo mismo con los contenidos (elementos de la lista 'allContent). Por último, marca como activa la pestaña seleccionada y muestra el contenido asociado a esta.

La tercera y última función principal, closeTab, cierra la pestaña y la elimina de la lista 'allTabs' en primer lugar, y hace lo mismo con su contenido, eliminándolo de la lista 'allContent'. Después abre la primera pestaña, y si no existe (ya que hemos cerrado la única pestaña abierta) oculta el bloque general 'general div' que quedaba como elemento de fondo.

Atendiendo a las especificaciones de la aplicación, la función closeTab debe parar el proceso asociado a esa pestaña y liberar los recursos. Sin embargo, esta funcionalidad se implementará m ás adelante, ya que de momento solo ha quedado definido el sistema de pestañas a nivel gráfico.

Para ilustrar en esta memoria el sistema de pestañas empleado hemos puesto como ejemplo el sistema de pestañas de colores desarrollado previamente a la aplicación, pero diseñado específicamente para ésta. Una vez

implementado, las modificaciones necesarias para adaptar el sistema por colores a nuestra aplicación han sido mínimas: a nivel de diseño, cambiar el texto de las pestañas y eliminar el botón de cierre de la pestaña de búsqueda.

5.1.4 - Motor de búsqueda

Una de las principales funcionalidades de la aplicación que tiene como objetivo este Trabajo es la función de búsqueda. A través de ella, podemos interactuar con otros usuarios de reTHINK, gracias a la funcionalidad Global Discovery desarrollada por otros miembros del proyecto reTHINK. Por tanto, el objetivo principal de nuestro motor de búsqueda es realizar una consulta y procesar la información recibida de tal manera que el usuario pueda utilizar los resultados para conectar con otros usuarios.

Global Discovery

Para hacer uso del sistema Global Discovery, lo único que tenemos que hacer es realizar una petición HTTP al servidor donde está alojado el sistema con la palabra clave de búsqueda, y éste devolverá un objeto del tipo JSON con la información requerida.

A la hora de realizar la petición HTTP, usaremos el objeto XMLHttpRequest con el método GET para recibir el objeto JSON sin necesidad de abrir una nueva pestaña en nuestro navegador. La URL introducida es "https://rethink.tlabscloud.com/discovery/rest/discover/lookup?searchquery=" en la cual se añaden al final las palabras de búsqueda entre comillas, separadas por el símbolo '+'.

Resultados de la búsqueda

El resultado de la búsqueda viene, como ya hemos indicado antes, en un objeto con formato JSON, el cual incluye uno o varios resultados de usuarios reTHINK, o únicamente información de la búsqueda en caso de que las palabras clave no devuelvan ningún resultado.

En la figura número 32 se representan los resultados de la búsqueda de las palabras clave 'Elias' en formato JSON. En primer lugar, sobre fondo amarillo, se ofrece información sobre la búsqueda en sí. Esta información siempre está disponible, incluso cuando la búsqueda no devuelve ningún resultado.

```
{
    "instanceID": "telekom1",
    "responseCode": 201,
    "searchString": "Elias",
    "results": [
        {
            "resultNo": 0,
            "instanceID": "telekom1",
            "hashtags": "T-Labs Telekom",
            "description": "Elias works with reTHINK.",
            "rethinkID": "76ShVFF8hYdy67mJcXcvsIcfj17MynrJu6eY59CDUAI",
            "headline": "Elias Solana",
            "domain": "rethink.tlabscloud.com",
            "contacts": "www.telekom.de",
            "hasrethinkID": "true",
            "hyperties": [{
                "url": "hyperty=hyperty%3A%2F%2Frethink.tlabscloud.com%2F115fec51-...",
                "userID": "uid=user:\\gmail.com%5C34solana",
                "media": "VIDEO",
                "provider": "Deutsche Telekom"
            }, {
                "url": "hyperty=hyperty%3A%2F%2Frethink-dev.tlabscloud.com%2F49a31cf6...",
                "userID": "uid=user:\\gmail.com%5C34solana",
                "media": "CHAT"
                "provider": "Deutsche Telekom"
            }
        ]
    }
  ]
}
```


Figure 32: Ejemplo de resultados de búsqueda en formato JSON

```
"hyperties": [
    [{
        "hyperty://rethink.tlabscloud.com/af7dc161-150a-4a66-9c6e-78a48398b7cc": {
            "resources": ["chat"],
            "dataSchemes": ["comm"],
            "descriptor": "hyperty-catalogue://catalogue.rethink.tlabscloud.com/.well-known/hyperty/GroupChatManager",
            "startingTime": "2017-04-06T14:56:21Z",
            "hypertyID": "hyperty://rethink.tlabscloud.com/af7dc161-150a-4a66-9c6e-78a48398b7cc",
            "userID": "user://gmail.com/34solana",
            "lastModified": "2017-04-06T14:56:21Z",
            "status": "live",
            "expires": 3600,
            "runtime": "runtime://rethink.tlabscloud.com/3faf0589-43bb-8df0-9512-e214baf6d55d"
        }
```


Figure 33: Objeto JSON que muestra los hypertys de un resultado con información adicional

Después se ofrece la lista de resultados retornados por la búsqueda. En este ejemplo, sólo se devuelve un resultado. Sobre fondo naranja, hay 10 campos de datos de cada resultado, dando principalmente información sobre el usuario. El último campo, 'hyperties', devuelve la lista de hypertys activos que tiene la cuenta de ese usuario, incluyendo el identificador único URL con el que podemos establecer conexión (en la filosofía reTHINK es el único dato que necesitamos).

En futuras versiones del sistema Global Discovery se ofrecen todos los hypertys utilizados por el usuario en las últimas 2 horas, aunque en el momento de la búsqueda no estén siendo utilizados (ver figura número 33). Por tanto, hay que discriminar entre los hypertys activos o no (por ejemplo, no se podría hacer una videollamada si el usuario no está activo en ese momento). En cualquier caso, las modificaciones necesarias en el código son menores, por tanto el formato expuesto en la figura número 32 sirve como referencia de cara a la implementación del motor de búsqueda en nuestra aplicación.

Teniendo en cuenta el formato del objeto de resultados del tipo JSON y la información que éste contiene, comose describe en la sección Arquitectura del desarrollo la decisión de diseño es definir el proceso de búsqueda en 3 fases o elementos: Barra de búsqueda, muestra de resultados (sólo usuarios), y muestra detallada de un usuario.

Barra de búsqueda

Figure 34: Barra de búsqueda

Cuando arrancamos la aplicación, el primer elemento que se muestra por pantalla es la barra de búsqueda. Como se puede ver en la figura número 34, la barra de búsqueda tiene 4 elementos: el logotipo de reTHINK, el cuadro de texto donde se introducen las palabras de búsqueda, el botón de búsqueda y un cuadro que presenta, al principio de la aplicación la resolución utilizada, y cuando el usuario se ha conectado satisfactoriamente cambia el text a 'Connected'.

Para ejecutar la búsqueda con las palabras introducidas en el cuadro de texto, se debe hacer click en el botón de búsqueda con la palabra 'Search', el cual ejecuta la función 'createSearchTab()'. Como funcionalidad adicional, también se puede realizar pulsando la tecla 'Enter'.

Debido a que esta barra de búsqueda aparece en cualquier caso y desde el inicio de nuestra aplicación, se puede definir de manera estática en lenguaje HTML (en el archivo index.html, ver figura número 19), lo cual simplifica significativamente su implementación. Esto no será posible en el resto de elementos de búsqueda, como la muestra de resultados.

Muestra de resultados

Como se ha descrito anteriormente, una vez pulsado el botón de búsqueda (o la tecla 'Enter') pasa a ejecutarse la función 'createSearchTab'. A partir de este momento, la operación de muestra de resultados se desarrolla en el archivo search.js (ver figura número 19), mostrando esos resultados en una nueva pestaña.

En primer lugar, creamos esa nueva pestaña de búsqueda dentro de la función 'createSearchTab', y para ello usamos de manera íntegra el código desarrollado en el sistema de pestañas, en la función 'createTab'. En el caso de que ya hayamos efectuado una búsqueda con anterioridad, se abrirá la pestaña de búsqueda ya utilizada en vez de abrir una nueva.

Una vez creada la pestaña (aún vacía), ejecutamos la función 'send-

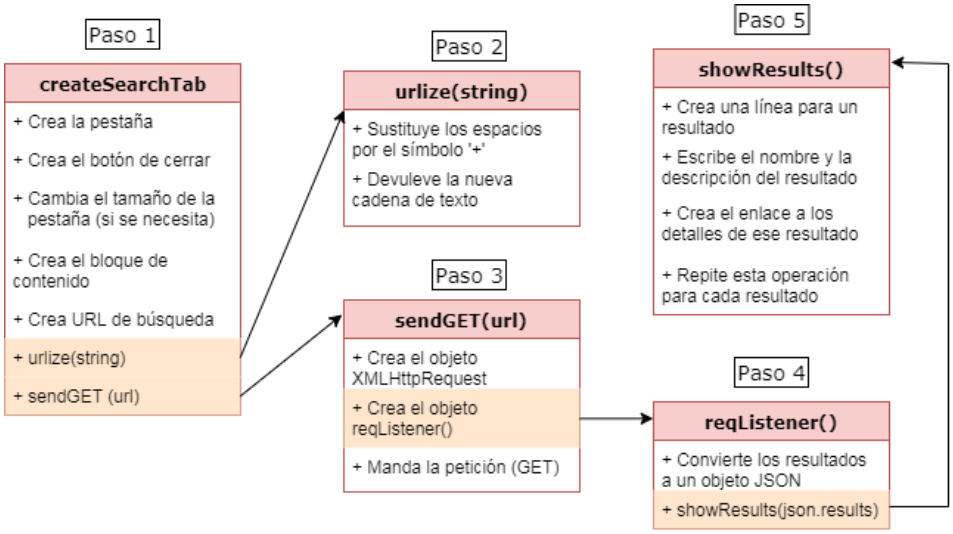


Figure 35: Proceso de búsqueda y muestra de resultados

GET' introduciendo como parámetro la URL de búsqueda (*https:// rethink.tlabscloud.com/discovery/rest/discover/lookup?searchquery=* mas las palabras de búsqueda separadas por el símbolo '+'). En la función sendGET, el objeto XMLHttpRequest efectuúa una petición al servidor del sistema Global Discovery usando la URL de búsqueda, y permanece a la escucha de respuesta.

Antes de enviar la petición, la función auxiliar 'urlize' asegura que la URL de búsqueda es válida. Una vez enviada, la función reqListener recibe la respuesta del servidor, y la convierte en un objeto JSON (ya que la respuesta es recibida en una cadena de text JavaScript). Inmediatamente después ejecuta la función 'showResults', que recibe como parámetro el objeto JSON que contiene los resultados de la búsqueda. En la figura número 35 se describen las funciones que intervienen en este proceso y su orden de ejecución.

La función 'showReults' se encarga de, una vez recibidos los resultados en formato JSON, presentar al usuario por pantalla esos resultados como se muestra en la figura número 36. Esta función recorrerá cada uno de los resultados, e imprimirá por pantalla una línea para cada uno de ellos, en la cual se mostrará el nombre del usuario y una pequeña descripción.

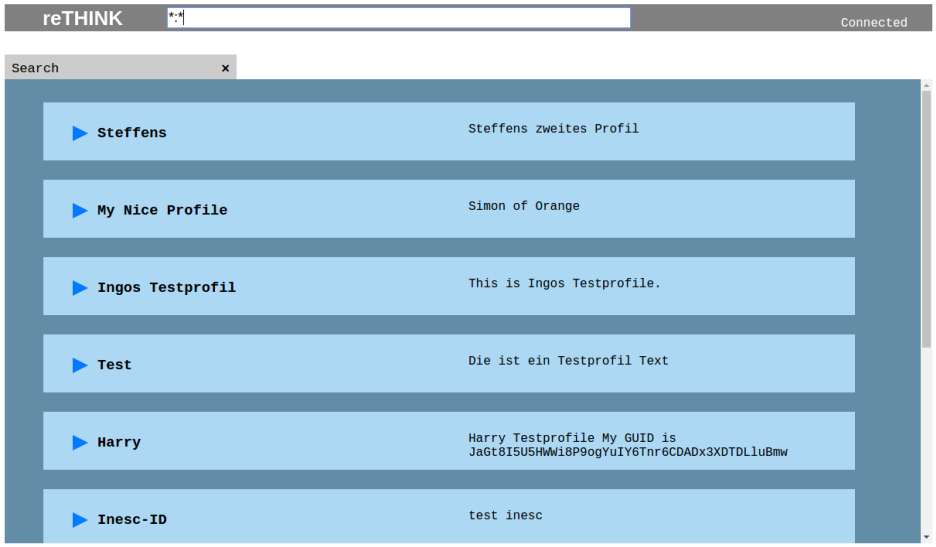


Figure 36: Muestra de resultados

Por último, la función 'showResults' crea un enlace al objeto de datos de cada resultado (información adicional y hpyertys activos, ver figura número 32), al cual se puede acceder haciendo click sobre el nombre o la descripción de ese resultado (en la figura número 36, sobre el recuadro azul claro).

Muestra de un usuario

Como se indica anteriormente, si se hace click sobre un resultado, se muestran los detalles de éste. La función 'buildModal' se encarga de ejecutar este proceso, abriendo una ventana modal con toda la información de ese resultado y la posibilidad de conectar con algún servicio que utilice el usuario reTHINK de ese resultado (ver ejemplo en la figura número 37).

En la nueva ventana modal, en la parte superior se muestra la información del usuario y la posibilidad de conectar con él. A la izquierda se muestra, o bien un icono genérico (como en el caso de la figura número 37) o el icono de la identidad del usuario(por ejemplo, el icono de la cuenta Gmail). En el bloque del medio se muestran los detalles de contacto, y en la parte derecha se muestra un icono sobre el que haciendo click podemos abrir una ventana de chat con ese usuario. En el caso de que el usuario con el que queremos conectar no tenga el hyperty de chat activado, este icono

Figure 37: Muestra en detalle de un resultado de una búsqueda

no aparece visible (en futuras versiones aparece 'Not connected').

Por último, en la parte inferior se muestran el resto de hypertys que utiliza este usuario. Esta funcionalidad no está implementada en esta aplicación, quedando espacio libre para un posible futuro desarrollo.

5.2 - Creación de un servicio de comunicación para la aplicación

En esta fase el objetivo principal es crear un servicio de comunicación basado en hypertys (microservicios) ya desarrollados y desplegarlo en nuestra aplicación en una nueva pestaña haciéndola plenamente funcional. Para este servicio usaremos dos hypertys ya desarrollados y disponibles en el repositorio oficial de reTHINK en GitHub: chat y videollamada (DTWebRTC). A nivel gráfico (front-end), el desarrollo lo haremos desde cero, usando los lenguajes de programación HTML, CSS y JavaScript.

Durante el proceso de desarrollo, el primer paso será crear la interfaz gráfica necesaria para este servicio que integra chat y videollamada. Debido a que los elementos deben ser creados de manera dinámica (y no de forma

estática al principio de la ejecución), el código necesario estará escrito en un fichero JavaScript aparte llamado 'videochat.js', como se muestra en la estructura de archivos descrita en la figura 19.

Después pasaremos a estudiar el código de los hypertys a utilizar, que funciones ofrecen y, una vez seleccionadas, integrarlas en el archivo 'videochat.js' para hacer la aplicación plenamente funcional.

5.2.1 - Diseño e implementación de la interfaz gráfica

El primer paso en el desarrollo del servicio de comunicación es el diseño e implementación de los elementos gráficos necesarios para la aplicación. Pero durante la ejecución, antes debemos crear una nueva pestaña (con el sistema de pestañas descrito con anterioridad) que incluya a estos elementos que proporcionan el servicio de chat y videollamada. Dentro de la función 'createVideoChatWindow' crearemos todo lo necesario (ver figura número **??**).

Una vez creada la nueva pestaña, pasamos a construir la intefaz del servicio de comunicación. Atendiendo al diseño básico propuesto en la sección Arquitectura del desarrollo, el diseño a nivel gráfico final es el expuesto en la figura número 38.

En ella, en la parte superior se expone el esquema de elementos gráficos cuando se usa únicamente el chat, en la parte media el esquema cuando se usa el chat mas la videollamada, y en la parte inferior cómo los elementos se organizan en el código. Todos ellos están marcados con la etiqueta HTML en la cual están codificados. Para ver el resultado final de éste diseño, se pueden observar las figuras número **??** y **??**. A continuación pasamos a describir los elementos en detalle.

En la primera fila, llamada row1, el único elemento es una línea de texto (elemento HTML del tipo H3) que contiene el nombre del usuario con el que se ha establecido la conexión. Ésta es la única columna que no cambia a lo largo del uso del servicio de chat + videollamada, por lo que a partir de este punto haremos una distinción entre el caso de chat y el caso de chat + videollamada.

Escenario de uso del chat

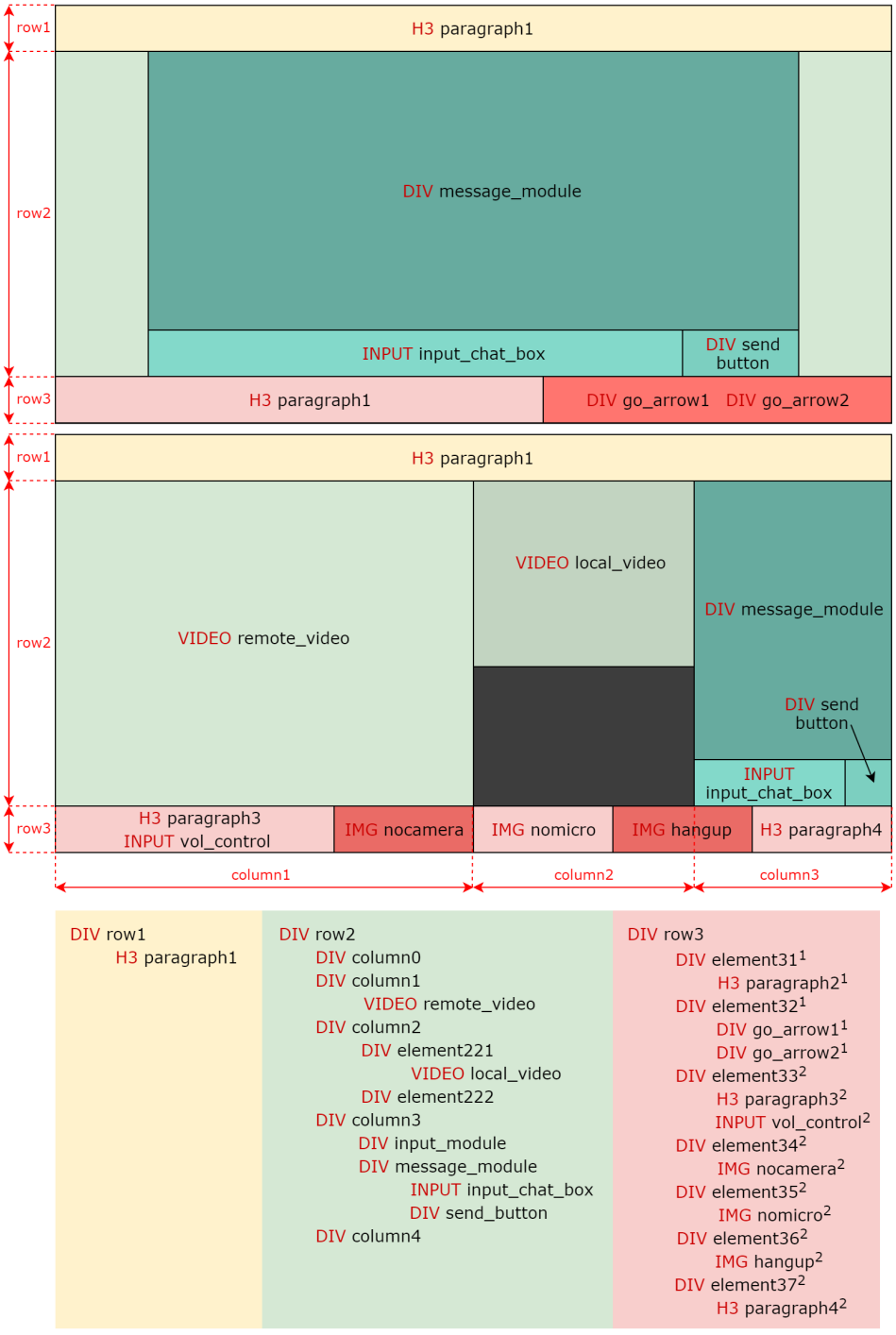


Figure 38: Esquema de los elementos gráficos de chat y videollamada

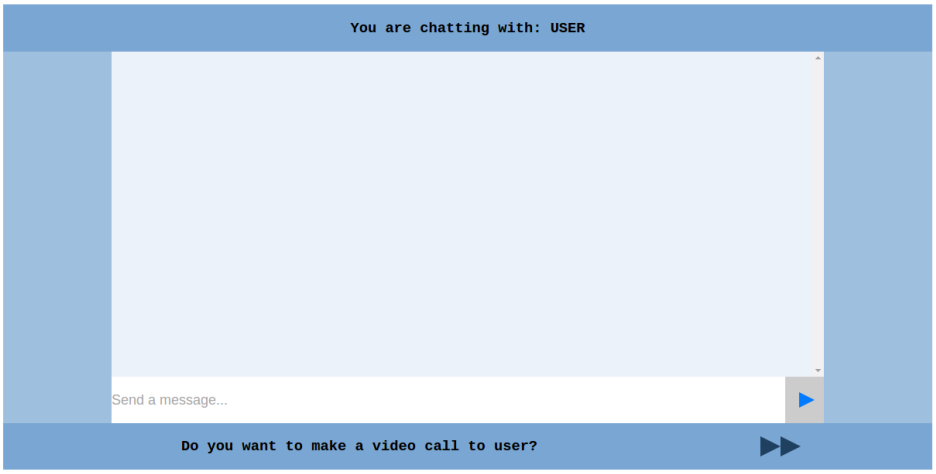


Figure 39: Pantalla de chat

En la segunda fila, llamada row2, tendremos una única columna funcional (column3, ver figura número 38) visible en la pantalla, la cual contiene 3 elementos: 'message module', bloque en el que aparece la conversación con el otro usuario del chat, 'input chat box', cuadro de texto donde introducimos un nuevo mensaje a enviar, y 'send button', botón que envía el texto introducido en el elemento anterior. También se puede enviar el texto pulsando la tecla 'Enter'. Además de la columna 3, 'column0' y 'column4' proporcionan márgenes a la izquierda y a la derecha respectivamente.

En la tercera fila, llamada row3, aparecen dos bloques distintos: el primero, situado más a la izquierda, contiene una línea de texto cuyo contenido ofrece la posibilidad al usuario de empezar una videollamada, y el segundo contiene a su vez dos elementos en forma de flecha sobre los cuales, si el usuario hace click con el cursor, manda una invitación al otro usuario para empezar una videollamada (para observar su implementación final, ver figura número **??**). Asimismo, si cualquiera de estas dos flechas es pulsada, se despliegan el escenario de chat + videollamada (en la figura número 38, el bloque en el medio) automáticamente, sin esperar a la respuesta de la invitación por parte del otro usuario.

Escenario de uso del chat + videollamada

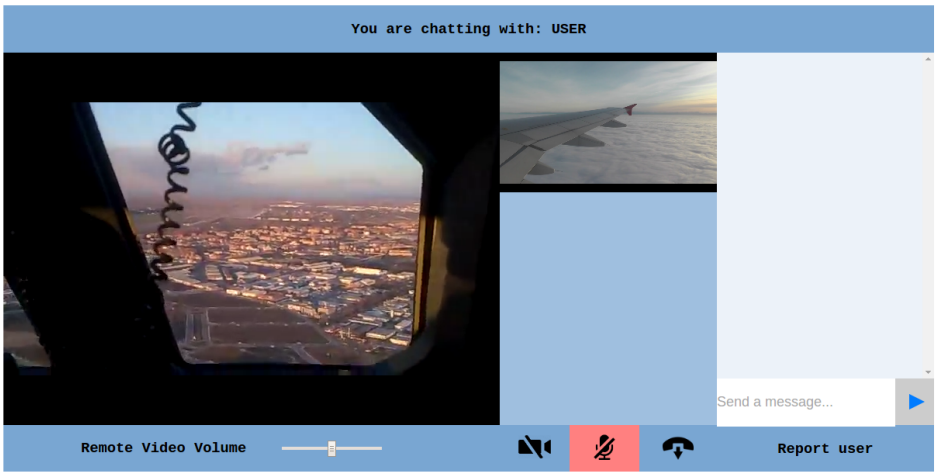


Figure 40: Pantalla de chat y videollamada

En la segunda fila, llamada row2, tendremos 3 columnas, todas ellas funcionales. En este escenario de uso, los bloques column1 y column2 aparecen en pantalla, ya que estaban ocultos en el escenario de chat (ver figura número 38). En la parte derecha permanece la columna asociada al chat, column3, la cual ve reducida su espacio a menos de la mitad para hacer hueco a los elementos de vídeo. En la parte central, column2 contiene el vídeo recogido por la cámara del usuario (local video), mas un espacio libre (destinado en el futuro a información adicional sobre el otro usuario). Y en la parte izquierda, ocupando la mitad de la pantalla, column1 contiene el vídeo recogido por la cámara del otro usuario con quien establecemos la conexión (remote video).

En la tercera fila, llamada row3, los 2 bloques que aparecían en el escenario de chat quedan ocultos, pasando a tener 5 bloques distintos. De izquierda a derecha, en primer lugar tenemos un bloque de control de volumen, el cual contiene una línea de texto (paragraph3) y un elemento deslizador que selecciona el volumen (vol control). Después se encuentran tres iconos de configuración de la videollamada: el primero (nocamera) activa o desactiva la cámara local, el segundo (nomicro) activa o desactiva el micrófono local), y el tercero (hangup) concluye la llamada, volviendo a la pantalla del escenario de chat. El último elemento, paragraph4, es simplemente una línea de texto.

Funciones auxiliares

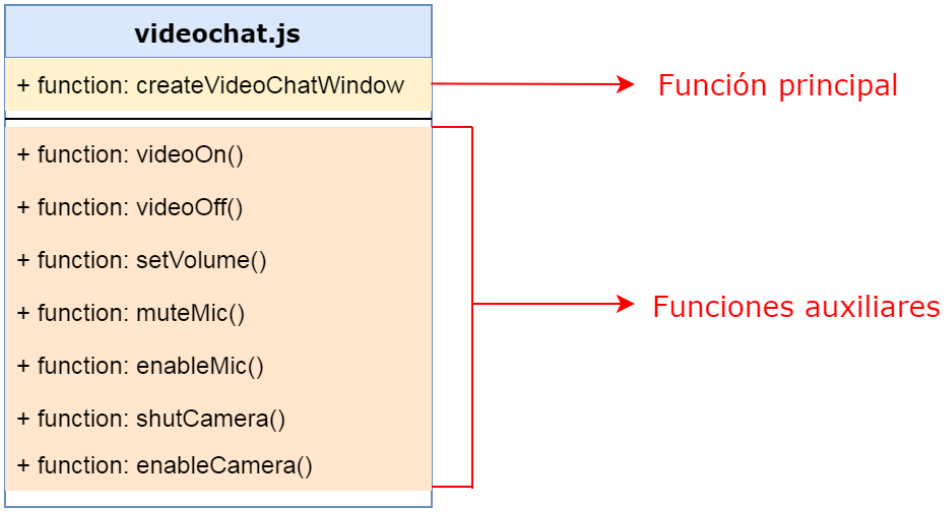


Figure 41: Funciones asociadas a la interfaz gráfica del archivo videochat.js

Como se menciona al principio de esta sección, la función createVideoChatWindow realiza todas las acciones hasta ahora descritas. Sin embargo, hay un elevado número de elementos gráficos (iconos, cuadros de vídeo, deslizadores) que interactúan con la aplicación. Las funcionalidades de estos elementos las definen 7 funciones implementadas al final del archivo 'videochat.js', cuyo listado se puede ver en la figura número **??**.

Las dos primeras, videoOn y videoOff, son las encargadas de modificar, ocultar y fijar los elementos gráficos para pasar del escenario de sólo chat al escenario de chat y videollamada. Actúan cuando se invita a un usuario a realizar una videollamada y cuando ésta se termina (accionada desde el icono 'hangup'), pero también cuando aceptamos la invitación de otro usuario o éste termina la videollamada.

La función setVolume se encarga de controlar el volumen del vídeo recibido por la cámara del otro usuario según la información recogida por el deslizador 'vol control'.

Por último, las funciones 'muteMic', 'enableMic', 'shutCamera' y 'enableCamera' se encargan de apagar o encender la señal enviada del micrófono y la cámara del usuario (local) hacia el otro usuario (remoto).

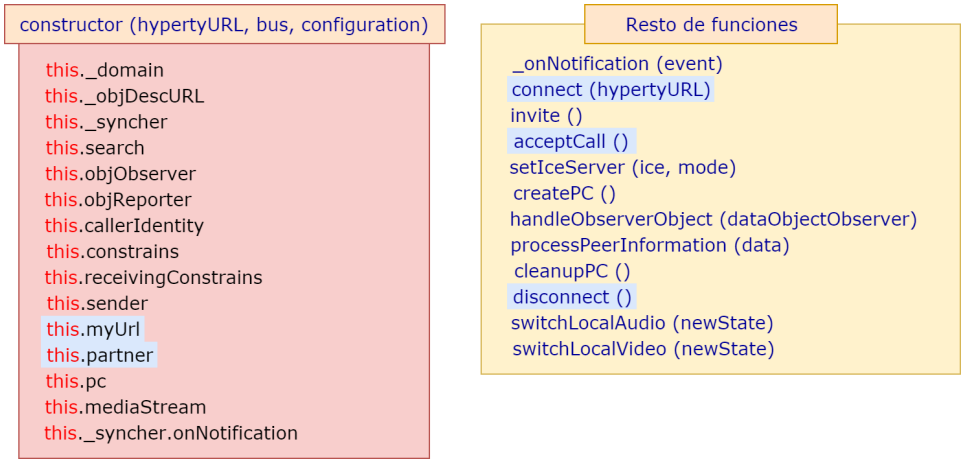


Figure 42: Estructura del código del hyperty de videollamada (DTWebRTC)

5.2.2 - Implementación de los hypertys en la aplicación

Una vez desarrollada la interfaz gráfica, utilizaremos los hypertys de chat y de videollamda para hacer el servicio de comunicación de nuestra aplicación plenamente funcional. El código fuente de los hypertys está alojado en la carpeta 'catalogue objects' (ver figura número 28), y éstos archivos ya están instalados en ese lugar desde la primera configuración del entorno de desarrollo, por lo que podremos usar sus funcionalidades directamente.

Hyperty de videollamada

En primer lugar haremos un estudio del funcionamiento de los hypertys. En el caso del hyperty de videollamada (DTWebRTC), el código contiene un constructor y distintas funciones que se pueden aplicar sobre un objeto de este tipo. En la figura número 42 se pueden ver, estando marcados en azul los atributos y las funciones de las que haremos uso directamente en nuestra aplicación. El resto de ellas, o bien no son necesarias para nuestra aplicación o son utilizadas de manera interna por el hyperty.

Volviendo a nuestra aplicación, primero declaramos las variables necesarias a lo largo de nuestra aplicación, entre ellas y la más importante, la variable que contiene al hyperty mismo a lo largo de nuestro código. El sigu-

Figure 43: Carga del hyperty de videollamada en la Hyperty Runtime local

iente paso será cargar el hyperty en el Hyperty Runtime, el cual a través del Catálogo dará valores a los atributos del hyperty (ver atributos en la figura número 42). Una vez realizado este paso, nuestra aplicación está lista para efectuar una videollamada.

Después de inicializar el hyperty en nuestra aplicación, el siguiente paso será declarar los escuchadores (EventListener), en nuestro caso dentro de la función 'hypertiesLoaded'. La función de estos elementos es recibir cambios en el estatus de la llamada y proceder en consecuencia. Los escuchadores definidos son cuatro: 'incomingcall', 'localvideo', 'remotevideo' y 'disconnected' (ver figura número 44).

El escuchador 'incomingcall' se activa cuando recibimos una invitación por parte de un usuario, y en caso de ser aceptada la invitación, desencadena todo el proceso de establecer una llamada. La primera acción que este escuchador provoca es la aparición de una ventana modal (ver figura número **??**) que acepta o rechaza la invitación. Si la invitación es aceptada, este hecho es notificado al usuario que ha mandado la invitación usando el método 'acceptCall' (ver figura número 42) para a continuación crear una nueva pestaña de chat y videollamada como la de la figura número **??**.

Los escuchadores 'localvideo' y 'remotevideo' proporcionan a la aplicación los recursos necesarios para poder reproducir la cámara de los dos usuarios. Y por último, el escuchador 'disconnected' recibe la señal de que el otro usuario ha abandonado la conversación, por tanto cierra la ventana de videollamada para volver de nuevo a la ventana de chat.

Por último, para el caso en el que el usuario local quiera iniciar una videollamada, sólo quedaría asignar al botón de establcer una videollamada

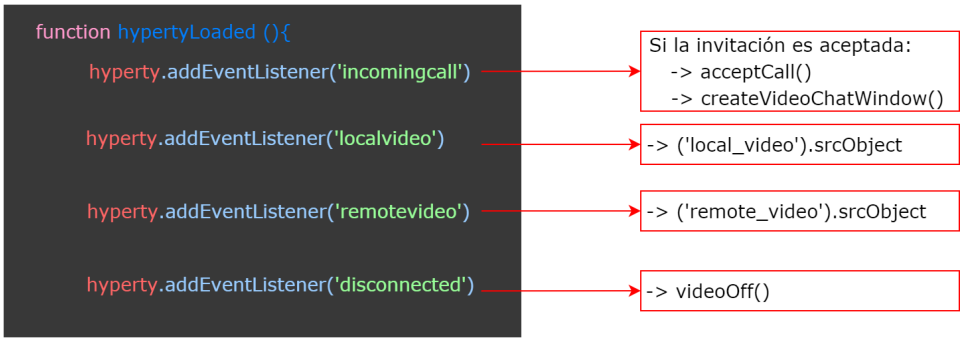


Figure 44: Lista de elementos EventListener (escuchadores) presentes en la aplicación

(en la figura número 38, los iconos 'go arrow1' y 'go arrow2') el método 'connect', dándole como parámetro la URL del hyperty con el que queremos establecer la conexión. Éste método se aplicaría sobre nuestro hyperty local de videollamada, que desencadena el proceso de invitación al otro usuario.

Hyperty de chat

En el caso del hyperty de chat, su estructura de archivos es más compleja. En primer lugar, el hyperty de chat proporciona, por un lado conversación con varios usuarios (chat de grupo), y por otro lado mantener varias conversaciones a la vez. En nuestra aplicación sólo usaremos la segunda posibilidad.

Para conseguir estas características, el hypert de chat consta de 3 archivos diferentes: 'GroupChat', el hyperty que despliega un usuario de chat estándar, 'GroupChatManager', el hyperty que despliega el administrador del chat, y 'ChatController'. La separación de las funciones del hyperty en 'GroupChat' (o 'GroupChatManager') y 'ChatController' es la que permite mantener varias conversaciones a la vez desplegando un sólo hyperty: Un Hyperty del tipo 'GroupChat' tiene varios 'ChatController', y cada uno de estos últimos maneja una conversación.

Además, para conseguir mayor eficiencia durante la ejecución de la aplicación, durante el desarrollo se decide crear un nuevo archivo llamado 'chat.js' que contiene un prototipo del hyperty 'GroupChatManager'. La razón es la posibilidad de establecer varias conversaciones de chat a la vez: esto incrementa de manera potencial el número de objetos de chat creados, y por tanto

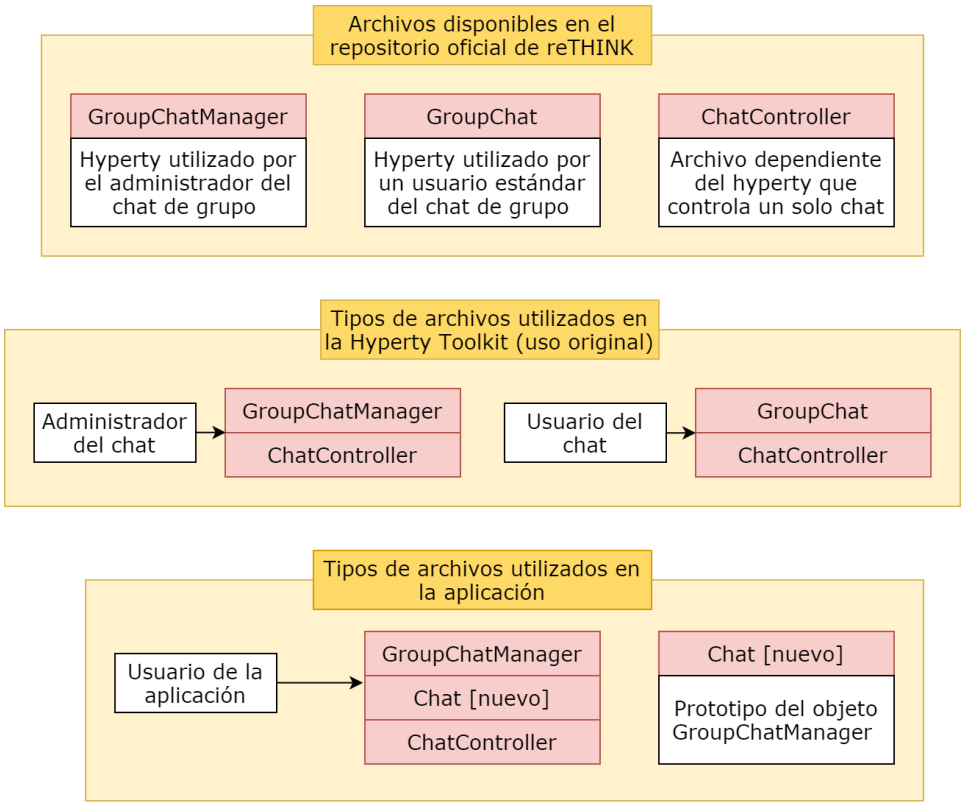


Figure 45: Lista de archivos relacionados con el servicio de chat

Figure 46: Carga del hyperty de chat en la Hyperty Runtime local

el número de accesos a las funciones relacionadas con el hyperty. Además, de esta manera podemos seleccionar y mantener en el código sólo las funciones que vayamos a utilizar. El archivo 'chat.js' se alojará en la carpeta 'src', según el esquema propuesto en la figura número 19

Por otro lado, emplearemos únicamente el hyperty 'GroupChatManager' en vez de ambos. La razón es que, según las especificaciones, nuestra aplicación ofrece un servicio de chat entre dos usuarios, en vez de un chat de grupo. Una vez realizadas estas consideraciones, pasamos a la implementación del hyperty de chat en nuestra aplicación, en el archivo 'videochat.js'.

Como ocurría con el hyperty de videollamada, el archivo del hyperty de chat 'GroupChatManager.js' se encuentra entre los objetos del catálogo guardado de manera local en Docker, por lo que podemos hacer uso de él directamente. Por tanto, el primer paso será declarar las variables iniciales en el archivo 'videochat.js'. En este caso, declaramos una variable llamada 'hypertyChat', donde guardamos el hyperty, y un array de objetos 'chats[]', en el cual almacenaremos las conversaciones.

El siguiente, de manera análoga a la videollamada, es cargar el hyperty en el Hyperty Runtime. En la figura número 46 se puede ver la implementación

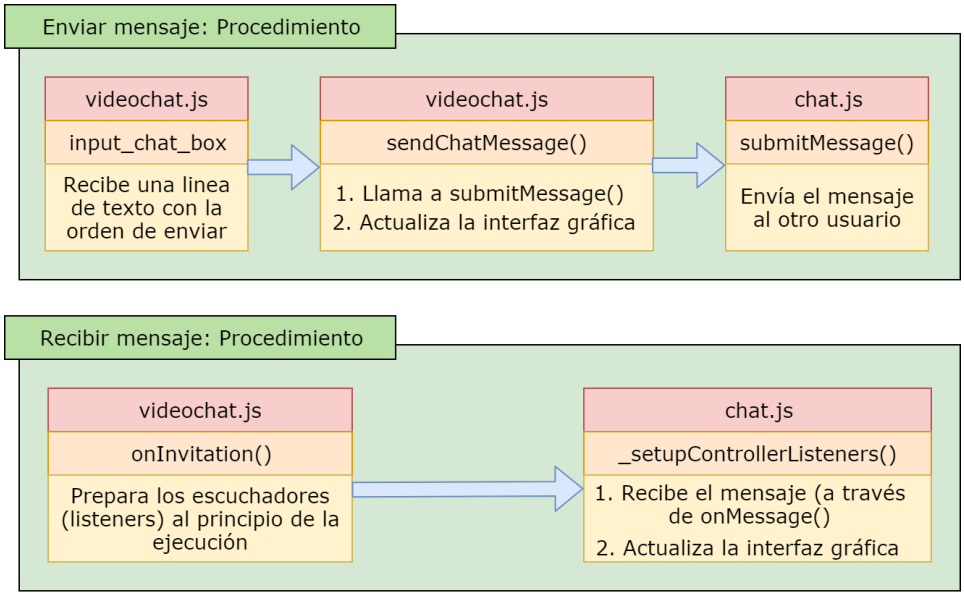


Figure 47: Procedimiento de intercambio de mensajes

en código: después de cargar el hyperty de videollamada, cargamos el hyperty de chat, incluyendo el caso en el que se recibe una invitación (en la figura se sustituyen ciertas líneas de código por comentarios por cuestiones de espacio)

A partir de aquí, la implementación se hace de distinta forma a la videollamada: las funciones del archivo 'Chat.js' tienen acceso y modifican el código contenido en 'videochat.js'. Por ejemplo, si se quiere modificar el cuadro de texto que contiene la conversación (ya que se ha enviado un nuevo mensaje), la función 'appendText' del archivo 'chat.js' podrá acceder al cuadro de texto contenido en el archivo 'videochat.js', y lo actualizará.

Las funciones del archivo 'chat.js' tienen acceso a estos elementos de la interfaz gráfica ya que sus referencias están guardadas como atributos en cada objeto del tipo 'chat': cada conversación tiene una referencia a su cuadro de texto o a su línea de entrada de texto, por ejemplo. En la figura número **??** se puede observar la lista de atributos y funciones de un objeto del tipo 'chat'.

En cualquier caso, no todas las modificaciones de la interfaz gráfica se

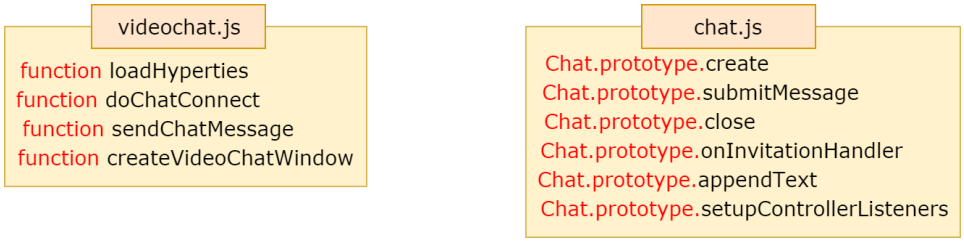


Figure 48: Lista de funciones relevantes en la función de chat (omitidas las relacionadas con la función de videollamada

hacen desde el archivo 'chat.js': en la figura número 47 se puede comprobar que durante el envío de mensajes la función 'sendChatMessage' es la que actualiza la interfaz gráfica, pero cuando se recibe un mensaje esta acción la realiza directamente la función que recibe el mensaje en el archivo 'chat.js', sin necesidad de enviar esta información al archivo 'videochat.js' donde está contenida la interfaz gráfica.

Quedando descritos los escenarios de envío y recepción de mensajes, sólo queda una funcionalidad básica por implementar (ya que no se consideran los escenarios de invitación de un nuevo usuario al ser una conversación entre dos): el envío de una invitación de chat. Para ello, desde el archivo 'search.js', en la ventana modal que se abre con la información de un usuario, al pinchar en el icono de mensaje se llama a la función 'doChatConnect' (localizada en el archivo 'videochat.js, ver figura número 48), la cual inicia el proceso de abrir una nueva conversación.

5.3 - Desarrollo de un nuevo 'hyperty' para la aplicación

En esta última fase de desarrollo el objetivo es crear un hyperty capaz de mantener varias videoconferencias y chats al mismo tiempo con distintos usuarios y desplegarlo en nuestra aplicación, para así mejorar la funcionalidad de la aplicación y completar los requisitos.

Como hemos visto anteriormente, el hyperty de chat tiene la capacidad de mantener varias conversaciones a la vez, no así el hyperty de videollamada, por lo que en esta sección nos centraremos en, tomando como base el hyperty de videollamada 'DTWebRTC', crear un nuevo hyperty que permita mantener varias llamadas a la vez, cumpliendo así los requisitos iniciales de

la aplicación.

Para ello, tomaremos como ejemplo el hyperty de chat: éste consiste en un hyperty con varios controladores, y cada controlador maneja una conversación. Esto conlleva que, por un lado el hyperty queda dividido en dos archivos distintos, y por otro lado el arcvhivo 'videochat.js' deberá guardar una instancia del hyperty y una base de datos con los distintos controladores activos (en el caso del chat, esta base de datos era un array de objetos).

A la hora de desarrollar este nuevo hyperty, empezaremos fuera de la aplicación siguiendo estos 3 pasos: el primero será separar el hyperty de videollamada en dos archivos distintos: el hyperty y el controlador. Para ello, intentaremos traspasar tantas funciones y atributos como sea posible del hyperty al controlador. El segundo paso será crear la base de datos de los controladores para hacer posible el manejo de varias conversaciones, y por último integraremos estos archivos en nuestra aplicación.

5.3.1 - Hyperty y controlador

Cuando se aborda la separación del hyperty en dos archivos, es fundamental tener en cuenta la compatibilidad del nuevo hyperty con las aplicaciones que lo usan, por tanto intentaremos crear unas funcionalidades que supongan mínimos cambios en el código de la aplicación. Esto significa que la aplicación no tendrá constancia de la existencia de controladores: la aplicación maneja un hyperty y distintas URL de hypertys destinatarios.

Por tanto, la aplicación jamás pedirá acceso o realizará cambios sobre los controladores: los cambios y las funciones las efectúa sobre el hyperty, proveyendo información a éste de la URL destino. Incluir esta URL destino como parámetro a la hora de realizar las funciones es la única modificación necesaria en el código de nuestra aplicación. Una vez tomada esta decisión de diseño, procedemos a separar el hyperty en un nuevo hyperty y su controlador, dividiendo los atributos y las funciones entre los dos archivos. Este proceso se puede ver en la figura número 49

Respecto a los atributos, algunos son comunes a todas las conversaciones del hyperty, pero el controlador necesita una referencia de ellas: estos son el dominio (domain), la URL propia (objDescURL) y el sincronizador (syncher), que es creado en el propio hyperty (this.syncher = new Syncher).

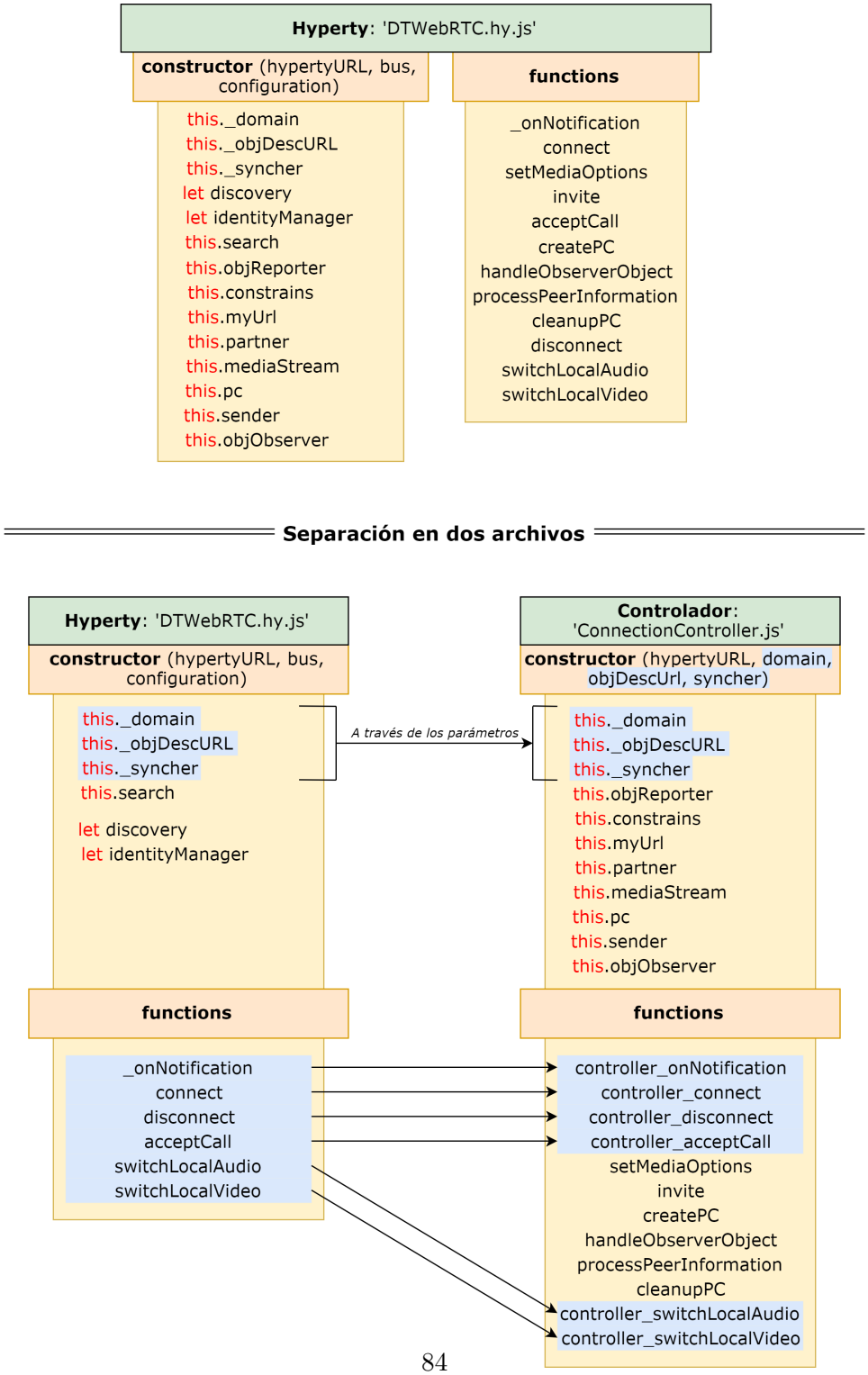


Figure 49: Separación del hyperty de videollamada en hyperty y controlador

Por tanto, cuando creamos un nuevo controlador, éste recibe estos tres parámetros en en su constructor, asegurando que estos tres atributos son los mismos en el hyperty y en el resto de controladores. El resto de atributos que no son distintos en cada controlador son transferidos directamente a éste.

En el caso de las funciones, todas ellas son transferidas al controlador, que es quien realizará las operaciones sobre la conexión. Sin embargo, atendiendo a las decisiones de diseño expuestas al principio de la sección, desde una aplicación externa se solicitará modificar un hyperty, no un controlador, proveyendo la URL del destinatario. Por ello, en el hyperty conservaremos las funciones de conexión y desconexión entre otras, las cuales recibiendo como parámetro la URL del destinatario ejecutarán la función deseada en el controlador que corresponde a dicha URL. La única labor de las funciones presentes en el hyperty es referenciar al controlador correspondiente (en la siguiente sección tomarán dos responsabilidades más).

La última modificación necesaria es importar la clase 'ConnectionController' (el controlador) en el archivo 'DTWerbRTC' (el hyperty) para que puedan operar entre sí.

5.3.2 - Uso y almacenamiento de los controladores

Habiendo separado el hyperty en dos archivos, hyperty y controlador, el propósito de esta sección es crear las condiciones para manejar varios controladores que puedan mantener varias conversaciones, siendo el principal reto la creación y manejo de una base de datos de los controladores.

Siguiendo el ejemplo del hyperty de chat, la primera aproximación es la creación en la aplicación (en el archivo 'vediochat.js') de una variable que contenga todos los controladores (en el caso del chat, un array de objetos). Sin embargo, esta idea es descartada desde el inicio ya que, como hemos indicado antes, uno de las carcacterísticas de este nuevo hyperty debe ser la compatibilidad con las aplciaciones externas que lo vayan a usar, y esta solución requiere cambios de gran calado en esas aplicaciones.

Por tanto, la solución tomada es la creación de un nuevo objeto intermedio entre el hyperty y el controlador: la base de datos de controladores. Este objeto tiene como atributos a todos los controladores, y a su vez este

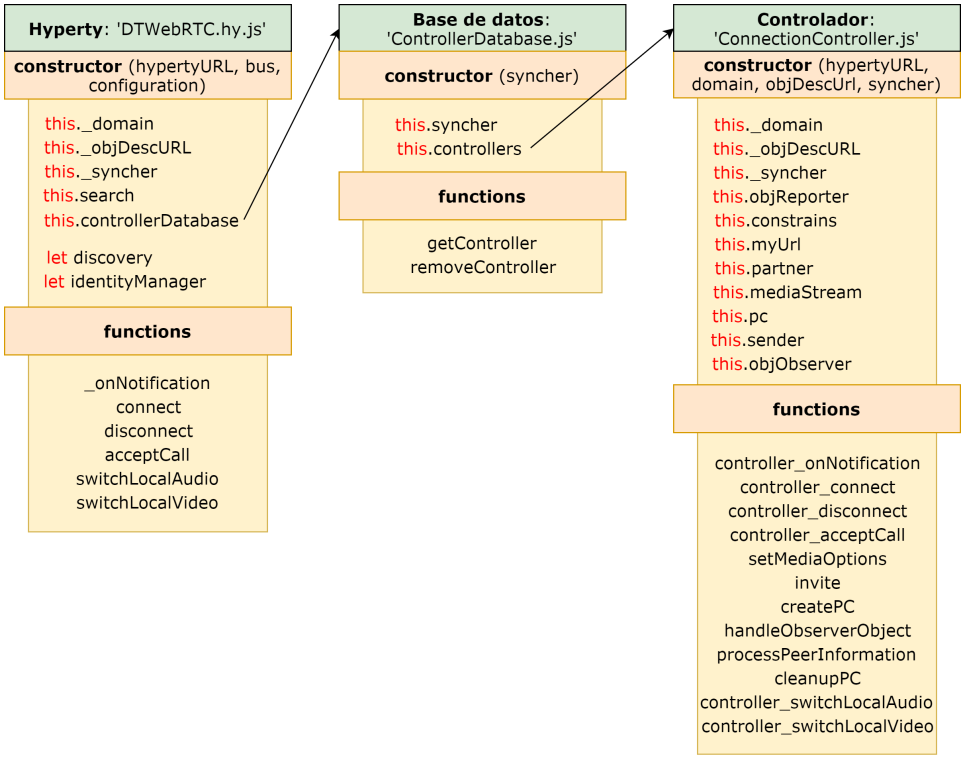


Figure 50: Estructura de código del hyperty de videollamada múltiple

objeto es un atributo del objeto hyperty.

El nuevo objeto 'ControllerDatabase' tiene como atributos un sincronizador (el común a todos los controladores de ese hyperty), y un objeto que guarda todos los controladores. Además, en el archivo 'ControllerDatabase.js' se definen dos funciones: 'getController' y 'removeController'. La función getController recibe una URL de destino y devuelve el controlador de la base de datos que maneja ese recurso, y si no existe crea uno nuevo. La función 'removeController' recibe una URL de destino y elimina el controlador asociado a ésta. Estos atributos y funciones se pueden observar en la figura número 50.

Estas dos funciones se usan en el hyperty (DTWebRTC.hy.js). La función getController se usa en las funciones 'onNotification', 'connect' y 'acceptCall', ya que por ejemplo, cuando creamos una nueva conexión con otro

usuario, este recurso debe ser guardado en un nuevo controlador, que es creado por la función getController. Esta misma función devuelve el controlador deseado para las funciones 'onNotification' y 'acceptCall', y la función 'removeController' elimina el controlador cuando la conexión ha terminado. De este modo, cualquier aplicación externa no necesita manejar los controladores: simplemente usuará el hyperty y la URL de destino.

Además, debemos modificar el código de los mecanismos de emisión y escucha de eventos (EventEmitter y EventListener). En el hyperty de videollamada original, éste está diseñado para sostener una sola conversación, por tanto estos mecanismos son funciones asociada al hyperty. Sin embargo, ahora debemos modificar el código para crear estos mecanismos en cada controlador, sin tener que modificar el cdoigo de aplicaciones externas. La solución tomada será crear una función llamada 'initListeners' en el hyperty (en el archivo 'DTWebRTCF.hy.js') que contiene los escuchadores (EventListener) y recibe mensajes desde los controladores, y los redirige hacia la aplicación externa, que recibe estos eventos desde el hyperty (y no desde los controladores).

Por último, debemos ajustar nuestro código, de manera extensiva en el hyperty y el controlador y levemente en la aplicación, para que estos nuevos elementos sean totalmente compatibles.

5.3.3 - Integración en la aplicación

La fase final de implementación de este nuevo hyperty es su despliegue en la aplicación. Como se ha señalado antes, una de las decisiones de diseño es construir el hyperty de manera cuasi-transparente para la aplicación, de tal modo que una aplicación que haga uso del hyperty de videollamada sólo necesite pequeñas modificaciones para emplear este nuevo hyperty de videollamada múltiple.

El principal cambio es la necesidad de añadir como parámetro la URL de destino en cada ocasión que la aplicación hace uso de funciones del hyperty, ya que aunque la aplicación no conoce de controladores, el hyperty necesita saber a qué controlador referirse cuando la aplicación gestiona una conversación. Por ejemplo, si la aplicación hace uso de la función 'connect' (ver figura número 42), ahora deberá pasar por parámetro la URL del destinatario.

También deberá modificar los escuchadores (EventListeners), ya que ahora éstos reciben como parámetro la URL del otro usuario con el que se establece la conexión.

Por último, estos nuevos archivos (el hyperty, la base de datos y el controlador) deberán ser incuidos en la carpeta 'catalogue objects' de Docker, sustituyendo el antiguo hyperty 'DTWebRTC'.

5.4 - Validación

Para validar el desarrollo realizado con la aplicación empleamos, el usuario número 1 la aplicación desarrollada y el usuario número 2 la 'Hyperty Toolkit'. La razón fue la laboriosidad de configurar un entorno idéntico en dos equipos diferentes. En cualquier caso, este escenario asegura compatibilidad en el caso de utilizar la aplicación en ambos lados ya que los hypertys usados en la aplicación y en la 'Hyperty Toolkit' son los mismos.

Debido a ello, a la hora de validar el servicio de hyperty + videollamada, en el lado de la aplicación este servicio está integrado, pero en el lado de la'Hyperty Toolkit' se deben abrir dos ejecuciones distintas (una para la videollamada y otra para el chat). Esto limita la simulación de ciertos casos de uso: por ejemplo, en el caso de que la aplicación reciba una videollamada, en esa pantalla el chat puede estar no disponible (ya que en la aplicación tener videollamada asegura tener el chat activo, pero en la 'Hyperty Toolkit' son totalmente independientes).

En cualquier caso, éste método de simulación de uso nos permitió avanzar rápidamente (ya que se podía probar cada cambio efectuado sobre el código en cuestión de segundos), así como una visión muy aproximada del funcionamiento real de la aplicación. Los casos de usuario probados son los siguientes:

- Búsqueda de un usuario: Iniciación de la aplicación, introducción de identidad del usuario, búsqueda de un usuario. Elementos a comprobar: funcionalidad de la barra de búsqueda y presentación de resultados.
- Iniciar conversación chat: A través de una búsqueda, establecer una conexión chat con otro usuario reTHINK. Elementos a comprobar:

Sistema de pestañas, funcionamiento de la interfaz de chat, envío y recepción de mensajes.

- Iniciar videollamada: Partiendo de la ventana de chat, realizar una videollamada con el mismo usuario. Elementos a comprobar: Envío y recepción de vídeo y audio, funcionamiento de la interfaz de videollamada (incluyendo los iconos de configuración), correcto funcionamiento del chat, y finalización de la llamada.

- Recibir invitación de conversación chat: Una vez introducida la identidad de usuario, establecer una conversación chat. Elementos a comprobar: Envío y recepción de mensajes.

- Recibir invitación de videollamada: Una vez introducida la identidad de usuario, establecer una videollamada a invitación de otro usuario. Elementos a comprobar: Funcionamiento de la ventana de invitación, envío y recepción de vídeo y audio.

- Chat múltiple: Establecer varias conversaciones chat a la vez en pestañas distintas. Elementos a comprobar: Envío y recepción de mensajes en todas las conversaciones abiertas.

- Videollamada múltiple: Establecer varias videollamadas a la vez en pestañas distintas. Elementos a comprobar: Envío y recepción de vídeo y audio en todas las videollamadas establecidas.

A continuación se describen los resultados obtenidos con los casos de usuario descritos anteriormente, agrupados por el servicio ofrecido.

Gestión de identidades

Presente al principio de la aplicación, este sistema es desplegado por el 'Hyperty Runtime'. Funciona de manera correcta en nuestra aplicación, tardando entre 15 y 30 segundos en completar el proceso.

Sistema de búsqueda

El sistema de búsqueda funciona correctamente. La barra de búsqueda es plenamente funcional (bien a través del botón de búsqueda o de la tecla 'Enter'), los resultados son representados en la interfaz ggráfica de manera correcta, y el icono de establecer conexión también funciona correctamente,

abriendo una ventana de chat.

Chat

El servicio de chat es capaz de enviar y recibir mensajes bajo cualquier escenario (enviando y recibiendo la invitación de chat), así como de actualizar su interfaz gráfica acorde a los eventos recibidos, manteniendo una y varias conversaciones distintas al mismo tiempo. También es plenamente funcional durante la videollamada. Por último, el sistema de pestañas funciona de manera correcta, liberando los recursos utilizados en el servicio de chat cuando ésta se cierra.

Videollamada

El servicio de videollamada envía y recibe vídeo y audio de manera correcta, pero no hemos podido medir el retardo con el que se recibe. Pueden establecerse más de una videollamada al mismo tiempo, y el mecanismo de recepción de invitación funciona de manera correcta (estableciendo una conexión sólo si la invitación es aceptada). Los iconos de encendido y apagado de la cámara y control de volumen funcionan correctamente, y los recursos utilizados son liberados cuando la videollamada se termina.

Capítulo 6: Conclusiones y trabajos futuros

Este capítulo incluye todas las conclusiones asociadas al diseño y al desarrollo realizado, así como del proyecto reTHINK en general.

En primer lugar, se describen las conclusiones del desarrollo previo. Despúes, se efectúa un análisis técnico de las características de la arquitectura reTHINK, cuya evaluación se realiza a través del desarrollo previo y del desarrollo de la aplicación. Por último se exponen las conclusiones generales acerca del proyecto reTHINK, su impacto económico y social y los trabajos futuros asociados al proyecto.

6.1 - Conclusiones del desarrollo previo

En el capítulo 4 se describe el desarrollo previo, que aunque no forma parte de la aplicación, sus conclusiones fueron útiles para el desarrollo de ésta. A traveés del desarrollo del hyperty 'Slider' se hace una primera aproximación al funcionamiento y a las distintas posibilidades que ofrece la arquitectura reTHINK. En primer lugar, y la más importante, que el mecanismo de sincronización es lo suficientemente flexible como para crear un sistema de conexión bi-direccional, con unos tiempos de respuesta muy reducidos.

También se ha demostrado la posibilidad de desplegar nuevos hypertys (creados por el usuario) en la 'Hyperty Toolkit', aunque de una manera un tanto laboriosa (ya que hay que seguir el formato de archivos de manera cuidadosa, comprobando la coherencia de los nombres de los archivos desarrollados para que funcione). En cualquier caso, aunque en este escenario la aplicación sólo requería de un hyperty, no sería posible desplegar en la 'Hyperty Toolkit' una aplicación que usase más de un hyperty.

6.2 - Evaluación de la arquitectura reTHINK

Los puntos principales a evaluar son los indicados en la sección Objetivos del Trabajo, en el cuál se indicaban distintos aspectos de la arquitectura reTHINK a evaluar. Son los siguientes:

Reusabilidad de hypertys como microservicios

Una de las nociones centrales de la arquitectura reTHINK es el uso de los hypertys como mircroservicios, y su posiblidad de ser reutilizados a lo largo del tiempo y por distintas aplicaciones. Podemos concluir que, en términos generales, esta reusabilidad se ha demostrado positivamente.

En el servicio de comunicación creado en nuestra aplicación, se utilizaban dos hypertys distintos: uno de chat y uno de videollamada. En primer lugar, se ha demostrado posible desplegar dos hypertys en un mismo Hyperty Runtime, utilizando una misma identidad, requisito indispensable para su funcionamiento. Además, estos hypertys pueden ser reutilizados a lo largo del tiempo: se ha podido crear una conversación de chat, cerrarla y volverla a abrir, usando en todo momento los mismos hypertys a ambos lados de la conexión.

La complejidad a la hora de implementar varios hypertys en un mismo servicio no es grande. En nuestro caso, se han implementado dos hypertys en un mismo archivo Javascript, con la única necesidad de añadir las líneas de código relacionadas con el Hyperty Runtime e importar ciertos archivos, pudiendo utilizar las funciones de los hypertys directamente. En cualquier caso, se recomienda la creación de especificaciones para utilizar las APIs de los hypertys, incluyendo las líneas de código que son necesarias añadir para hacer funcionar a los hypertys en el Hyperty Runtime.

Por tanto, se concluye que la teoría de hypertys como microservicios se ha demostrado realizable en la práctica, siendo útil para el desarrollo de nuevos servicios.

Sincronización

La fase en la que se utiliza el mecanismo de sincronización del objeto de datos de manera más extensiva es en la última parte del desarrollo, durante la creación de un nuevo hyperty. En ella, comprobamos la posibilidad de utilizar varios objetos de datos con un mismo sincronizador, lo que permite manejar distintas conexiones a la vez de manera muy eficiente. Asimisimo, las funciones que proporciona el mecanismo de sincronización son sencillas y fáciles de utilizar. La actualización del objeto de datos cuando se recibe un mensaje se hace de manera casi inmediata, existiendo muy poco retardo durante la transmisión.

Como punto a mejorar, se recomienda la existencia en el repositorio ofi-

cial de reTHINK de una guía de utilización del mecanismo de sincronización, incluyendo las funciones que éste ofrece y sus posibilidades de aplicación.

De manera general, se puede concluir que el mecanismo de sincronización de datos funciona de manera muy eficiente, y cumple los propósitos para los que ha sido diseñado.

Motor de búsqueda

El aspecto a evaluar es el funcionamiento y practicidad del motor de búsqueda 'Global Discovery' usado en nuestra aplicación. De manera general, el mecanismo de búsqueda funciona correctamente y proporciona la información necesaria al usuario. Sin embargo, identificamos varios aspectos a mejorar:

En primer lugar, la información de los hypertys de cada usuario. Hay varios datos innecesarios, como el campo 'descriptor' o 'dataScheme', que no son utilizados a la hora de establecer una conexión, y algún dato que falta, como el dominio donde se aloja el hyperty (necesario en el caso del chat). Considerando que el formato del objeto JSON es flexible, esto no supone un problema grande siempre que éste formato se pueda modificar.

Otro aspecto a mejorar es el número de hypertys desplegados por usuario que se muestran cuando se hace una búsqueda. En un principio, el objeto JSON sólo devolvía los hypertys activos, pero en desarrollos posteriores pasó a mostrar los hypertys activos durante las últimas horas, lo cual crea escenarios de incompatibilidad. Por las dos razones expuestas, es necesario mejorar en la estandarización del objeto JSON de resultados de la búsqueda.

Como aspecto positivo, las características de búsqueda de 'Global Discovery' (búsqueda de usuarios, servicios, aplicaciones) le confieren el potencial suficiente para dar servicio a una arquitectura del tipo reTHINK, en la que con un sólo motor de búsqueda no sólo se buscan usuarios, sino todo tipo de entidades presentes en reTHINK.

Protocolo 'on-the-fly'

Aunque durante la realización de éste proyecto no hemos profundizado en este protocolo (que define la comunicación entre hypertys), el desarrollo y validación de esta práctica nos ha permitido conocer de manera general

acerca del rendimiento de éste.

El esquema de comunicación entre hypertys a través de un objeto de datos funciona de manera eficaz durante la ejecución, generalmente no presentando ningún problema de conexión durante la ejecución. Como aspecto a mejorar, la carga de los hypertys y los protocolos a utilizar al principio durante el arranque de la aplicación es relativamente lenta, pudiendo sobrepasar los 30 segundos, aunque este tiempo de carga está también relacionado con la gestión de identidades.

Gestión de identidades

El último aspecto a evaluar es el mecanismo de gestión de identidades. Aunque la independencia de las entidades de los usuarios respecto a los proveedores de servicios es una de las características más importantes de la arquitectura reTHINK, éste no ha sido uno de los puntos centrales de este Trabajo.

En cualquier caso, el desarrollo de la aplicación nos ha permitido hacer varias consideraciones acerca de como funcionan las identidades. Como aspecto positivo, identificamos que exista la posibilidad de elegir distintas identidades para distintos hypertys dentro de la misma aplicación. Por otro lado, resulta laborioso y lento por parte del usuario la asignación de identidades a cada hyperty al principio de la ejecución. De cara a futuros desarrollos, se debería poder hacer una pre-configuración de identidades por parte del usuario, de modo que éste elija que identidades quiere asignar a que hyperty y estas identidades de desplieguen automáticamente.

Otros aspectos

Respecto al entorno de desarrollo de hypertys creado por reTHINK, Hyperty Toolkit, ésta presenta varias ventajas y distintas debilidades. En primer lugar, cabe considerar que esta herramienta está diseñada para desarrollar hypertys, no aplicaciones. Eso quiere decir que, si se crea un servicio que utiliza más de un hyperty, este servicio no se podrá desplegar en la Hyperty Toolkit. Por tanto, existe la necesidad de una segunda plataforma para desarrolladores que permita desplegar aplicaciones y no sólo hypertys.

Por otro lado, es una plataforma relativamente fácil de usar, y no requiere de grandes modificaciones a la hora de desplegar un nuevo hyperty. Además,

existe una extensa y detallada documentación disponible en el repositorio oficial de reTHINK en GitHub, que facilita enormemente el uso de la Hyperty Toolkit.

6.3 - Conclusiones generales

La conclusión general es que la arquitectura reTHINK cumple a nivel técnico los propósitos iniciales por los cuales el proyecto fue lanzado. Consigue desplegar una arquitectura basada en microservicios, y los principales aspectos técnicos (sincronización, gestión de identidades) son funcionales.

De éstos aspectos, el que considero más relevante es la gestión de identidades. Este sistema, que divide los plano de identidades y servicios, pudiendo seleccionar qué identidad se desea para cada servicio, tiene un gran potencial en un entorno en el que la privacidad de los datos cada vez cobra más relevancia.

A nivel de aplicación en el mercado, reTHINK cumple los propósitos iniciales, pero es posible que el entorno global sea diferente tres años después de su inicio, y las soluciones válidas para el año 2014 no lo sean para el presente. Las grandes compañías que ofrecen servicios web han seguido aumentando su cuota de mercado sobrepasando las limitaciones de interoperabilidad entre protocolos, y esto significa que es poco probable que la arquitectura reTHINK pueda aspirar a ser un elemento central en el mercado web de masas.

Sin embargo, ciertos aspectos técnicos de la arquitectura reTHINK, como la gestión de identidades, pueden ser interesantes para ciertas instituciones (por ejemplo, gubernamentales) que desean controlar el flujo de información de sus plataformas, en lugar de asignarle esta labor a una compañía privada.

También puede resultar interesante la compatibilidad entre protocolos que reTHINK ofrece, sobre todo para pequeñas empresas que desean ofrecer servicios web a las que les es difícil acceder al mercado. En cualquier caso, existe una dificultad en este campo, y es la necesidad de que un número grande de empresas estén interesadas en reTHINK para asegurar la compatibilidad. Ésto se podría conseguir introduciendo la arquitectura reTHINK en plataformas como las explicadas en la siguiente sección de Trabajos Futuros, dando a conocer al público la arquitectura reTHINK e introduciéndola como

estándar en ámbitos concretos.

6.4 - Impacto económico y social

En distintas secciones de esta memoria se describen distintos aspectos socio-económicos que el proyecto reTHINK (y por ende, la aplicación por la que tiene objeto este Trabajo) aspira a influir o cambiar, y cómo las herramientas desarrolladas influyen en este proceso. En esta sección haremos un repaso de éstos conceptos, analizando el posible impacto futuro que el proyecto puede tener sobre distintos ámbitos de la sociedad.

En el plano económico, la aportación más relevante es que reTHINK aspira a crear oportunidades para que las empresas de servicios web más pequeñas puedan competir en mercados en los que hasta ahora les era difícil competir. A nivel técnico, reTHINK cumple las expectativas de interoperabilidad descritas en los objetivos establecidos en sus inicios. Sin embargo, el uso de reTHINK como estándar plantea una gran dificultad: su implantación debe ser generalizada, y ello, además de ser un objetivo muy ambicioso, se puede demorar en el tiempo. En un entorno dinámico de grandes cambios como son las tecnologías de la información, una tecnología puede quedar obsoleta en cuestión de uno o dos años. Este razonamiento plantea serias dudas sobre la posibilidad de que reTHINK tenga un impacto relevante entre las pequeñas empresas de servicios web.

En el plano social, reTHINK tiene por objetivo el ofrecer mayor facilidad a la hora de que dos o más usuarios se comuniquen desde plataformas distintas, manteniendo altos estándares de seguridad y privacidad para esos usuarios. En el sector público, estos aspectos tiene gran importancia, pudiendo los organismos públicos controlar los datos personales de los ciudadanos de manera transparente y fiscalizada, sin tener que confiar en una tercera empresa. Proveyendo reTHINK un marco de comunicación seguro y transparente, los organismos públicos tienen la posibilidad de ofrecer plataformas de participación ciudadana, incluyendo a través de la red a colectivos sociales más reacios a la participación en el día de hoy.

Al ser reTHINK un proyecto abierto, todos los individuos, organizaciones o empresas interesadas pueden acceder al código desarrollados y los informes desarrollados para la Comisión Europea. Esto significa que, en el caso de que las empresas colaboradoras en el proyecto no prosigan con la

explotación del mismo, esto puede ser posible a través de un tercero, y las conclusiones pueden ser utilizadas a nivel académico.

Como aspecto secundario, reTHINK puede influir de manera indirecta en el medio ambiente. A través de éstas plataformas ciudadanas descritas anteriormente, reTHINK puede colaborar a nivel técnico en el desarrollo de aplicaciones que tengna por objetivo el ahorro de energía o el control de índices de polución en entornos urbanos. Esta posibilidad ya ha sido explorada por el proyecto 'Smarter Together' [20].

6.5 - Trabajos futuros

La arquitectura del proyecto reTHINK tiene como objetivo proporcionar una plataforma en la que ofrecer servicios basados en el contenido (televisión bajo demanda, comunicación entre usuarios, servicios IoT), y la aplicación desarrollada en este trabajo pretende comprobar la utilidad y funcionamiento de esta arquitectura cuando ofrece servicios de comunicaicón entre usuarios. Así, ésta aplicación puede ser utilizada en el futuro por proveedores de servicios de comunicación, como muestra de capacidades de la arquitectura reTHINK.

Dentro de este contexto, durante el desarrollo de esta aplicación Deutsche Telekom consideró la posibilidad de aplicar las herramientas y las conclusiones de esta aplicación en tres proyectos auspiciados por el programa europeo Horizon 2020: 'Grow Smarter' (Colonia) [21], 'Smarter Together' (Munich) [20] y 'mySMARTLife' (Hamburgo) [22].

Aunque con distintas prioridades, estos tres proyectos tienen como objetivo común la transformación de las ciudades europeas, avanzando hacia un modelo en el que gracias al uso de las tecnologías de la información más recientes, se mejoren aspectos como la movilidad, el impacto ambiental o la participación ciudadana en la vida comunitaria.

Deutsche Telekom participó en cooperación con las tres ciudades alemanas previamente mencionadas proponiendo un sistema de comunicación en dos niveles (entre ciudadanos, y entre ciudadanos y el gobierno de la ciudad) que permitiera la participación ciudadana en la vida social de la ciudad y su administración. En el apartado técnico, esta propuesta incluía los conceptos desarrollados por reTHINK (gestión de identidades, reutilización de

recursos, calidad del servicio) que podían ser de interés para esos proyectos, implementados en la aplicación desarrollada durante este Trabajo.

La aplicación serviría como muestra de las capacidades de la arquitectura reTHINK de cara a un desarrollo completo de una plataforma de participación ciudadana, que incluiría otras funcionalidades adicionales no incluidas en la aplicación.

En el momento de la finalización de éste Trabajo, estas tres propuestas seguían su curso, siendo la más avanzada la relacionada con el proyecto 'Smarter Together' para la ciudad de Munich.

Capítulo 7: Marco regulador

En la realización de éste Trabajo se ha tomado como referencia la legislación europea a la hora de asegurar que la aplicación desarrollada cumple los requisitos legales. Si bien el desarrollo de esta aplicación se produjo en Alemania, el alcance de los posibles desarrollos futuros es de ámbito europeo. Además, debido al desconocimiento del idioma, para la elaboración de esta memoria no se ha realizado un estudio de la legislación alemana. De manera adicional, en esta sección se menciona brevemente la regulación española.

Las regulaciones establecidas por la Unión Europea tienen carácter de obligado cumplimiento en todos los estados miembros, y en esta sección haremos uso de dos tipos de actos jurídicos del derecho europeo: los reglamentos y las directivas. El reglamento tiene como objetivo la intervención directa sobre el Derecho de las distintas naciones que forman la Unión Europea, y es aplicable de manera obligatoria e igual sobre todos los miembros. En cambio, la directiva no pretende la unificación del derecho, sino la aproximación de las distintas legislaciones nacionales, eliminando las contradicciones y estableciendo los mismos requisitos materiales [23].

En cuanto a la normativa relacionada con la protección de datos del usuario, a nivel europeo existen dos directivas y un reglamento al respecto. La primera, la directiva número 95/46/CE, garantiza la protección de las libertades y de los derechos fundamentales de las personas físicas, y, en particular, del derecho a la intimidad , en lo que respecta al tratamiento de los datos personales [24]. La segunda, la directiva número 2002/58/CE, regula la protección de datos en el sector de las comunicaciones electrónicas con arreglo a la directiva anterior [25]. Por último, el reglamento número 2016/679 establece las normas relativas a la protección de las personas físicas en lo que respecta al tratamiento de los datos personales y las normas relativas a la libre circulación de tales datos [26].

A nivel europeo existen otros tres reglamentos y una directiva relacionados con la industria de las tecnologías de la información que afectan a este Trabajo. Por orden de relevancia, el primero es el reglamento número 283/2014, que establece unas orientaciones para el rápido despliegue y la interoperabilidad de proyectos de interés común en el ámbito de las redes transeuropeas en el sector de las infraestructuras de telecomunicaciones [27]. El segundo es el reglamento número 1025/2016, referente a la normalización en el campo de las tecnologías de la información. De menor impacto en el

ámbito de este trabajo, existe el reglamento número 1083/2012, que regula la obtención de estacísticas relacionadas con la sociedad de la información [28], y la directiva número 2016-2102, orientada a la accesibilidad de los sitios web y aplicaciones para dispositivos móviles de los organismos del sector público [29].

En la legislación española existen dos leyes que afectan a los servicios web. Por un lado, la Ley Oficial de Protección de Datos (15/1999), debido a que reTHINK almacena datos de los usuarios, y por otro lado la Ley de Servicios de la Sociedad de la Información (34/2002), que regula principalmente las páginas web con fines comerciales.

Como nota adicional, dado que ni la aplicación ni ningún elemento de la arquitectura reTHINK almacena cookies, en la realización de este Trabajo no se ha hecho un estudio al respecto. En el caso de que sí se almacenaran, existe una legislación europea y una española (con arreglo a la europea) que regula esta práctica.

Capítulo 8: Presupuesto

En esta sección se describe un presupuesto estimado para la realización del proyecto. En el, hemos considerado los gastos de personal y los gastos en el equipamiento empleado (hardware). Debido a que el software utilizado es libre (sistema operativo Linux), no se considera ningún gasto en licencias de software.

En primer lugar, el cálculo de horas de trabajo totales es 756. Se considera una duración de proyecto de 6 meses, trabajando 21 días por mes de media, 6 horas de trabajo al día.

En cuanto a la categoría de personal, se estima que sólo se necesita un programador con conocimientos de programación web fundamentalmente, estimando un salario por hora aproximado de 20 euros. Éste salario de trabajador autónomo está influido por el hecho de ser una contratación a medio plazo (6 meses), incluyendo impuestos.

Personal				
Categoría	Nombre	Sueldo (por hora)	Tiempo trabajado (en horas)	Total
Programador	Elías Solana Ortigosa	20.00 €	756	15,120.00 €

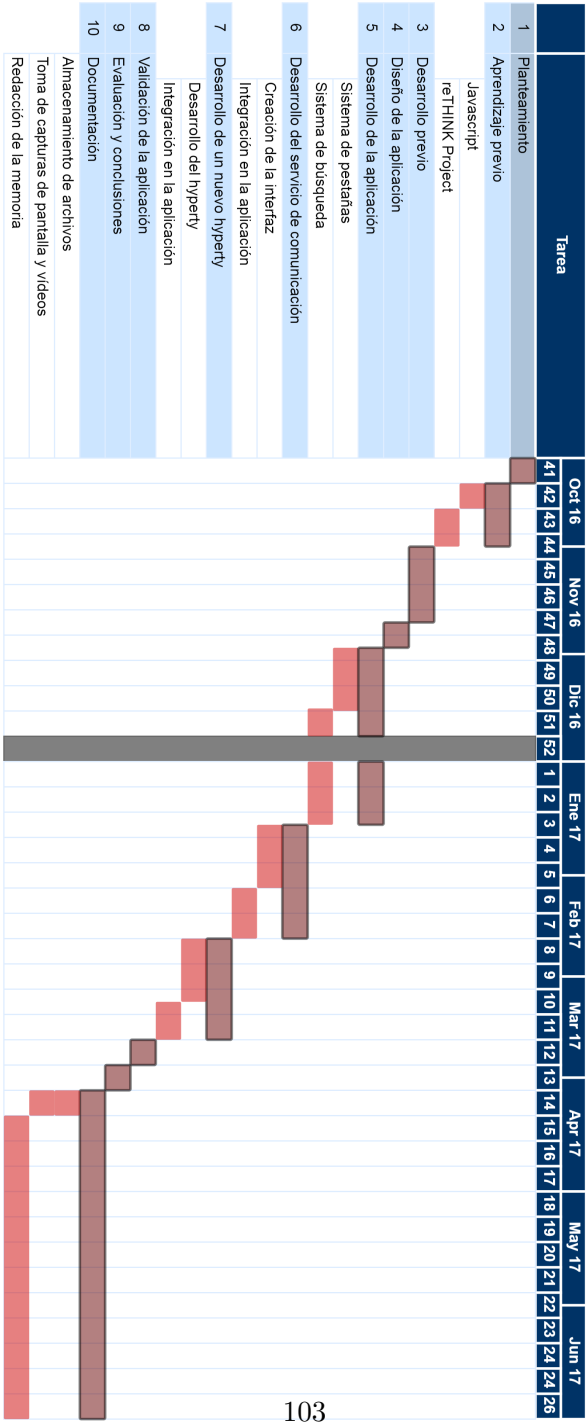
En la categoría de equipos se incluye el ordenador portátil y su monitor, el router y el servidor utilizado para almacenar la aplicación y todos los elementos necesarios para su desarrollo y validación. Este servidor tiene un precio de 0.095 € la hora, siendo utilizado el tiempo de trabajo calculado anteriormente. Para la elaboración de esta tabla se ha tenido en cuenta el precio inicial, su periodo de amortización (como tiempo de vida del elemento) y el uso dedicado a este proyecto.

Equipo (hardware)				
Descripción	Coste total	Uso destinado al proyecto	Periodo de amortización	Coste imputable
PC Portátil Fujitsu Lifebook T900	1,360.00 €	100%	48 meses	170.00 €
Monitor Iiyama B2480HS-B2	214,56 €	100%	48 meses	26.82 €
Router Linksys B2480HS-B2	55.44 €	100%	72 meses	4.62 €
Servidor dedicado AWS EC2	71.82 €	50%	-	35.91 €

La tercera y última tabla considera los costes totales de las anteriores categorías y el coste total del proyecto, impuestos incluidos.

Coste total	
Categoría	Coste
Gastos de personal	15,120.00 €
Gastos en equipo (hardware)	237.35 €
Total del proyecto (IVA incl.)	15,357.35 €

Apéndice: Diagrama de Gantt



Bibliografía

[1] I. Godlovitch et al., "Over-the-top (otts) players: Market dynamics and policy challenges," 2015.

[2] "rethink project - about," May 2017. https://rethink-project.eu/about/.

[3] "rethink project partners," May 2017. https://rethink-project.eu/about/partners/.

[4] "What is horizon 2020? - horizon 2020 - european commission," Jul 2017. https://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020.

[5] B. Familiar, *Microservices, Iot, and Azure: leveraging DevOps and microservice architecture to deliver SaaS solutions*. Apress, 2015.

[6] D. Jones, *JAVASCRIPT: novice to ninja*. O'Really Media, 2017.

[7] B. Smith, *Beginning JSON*. Apress, 2015.

[8] C. Coremans, *HTML: A Beginners Tutorial*. Brainy Software Corp., 2015.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," 1999.

[10] D. Ristic, *Learning WebRTC: develop interactive real-time communication applications with WebRTC*. Packt Publishing, 2015.

[11] H. A. (hta@google.com), "Google release of webrtc source code," Jul 2017. http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html.

[12] "Real-time communication in web-browsers (rtcweb)," Jul 2017. https://datatracker.ietf.org/wg/rtcweb/about/.

[13] "Webrtc 1.0: Real-time communication between browsers," Jul 2017. http://w3c.github.io/webrtc-pc/.

[14] S. Loreto and S. P. Romano, *Real-time Communication With Webrtc Peer-to-peer in the Browser*. Oreilly & Associates Inc, 2014.

[15] "Interactive connectivity establishment (ice): A methodology for network address translator (nat) traversal for offer/answer protocols," Jul 2017. https://tools.ietf.org/html/rfc5245.

[16] "What is npm?," Jul 2017. https://docs.npmjs.com/getting-started/what-is-npm.

[17] F. Santacroce et al., *GIT: mastering version control*. Packt Publishing, 2016.

[18] "What is docker," Jun 2017. https://www.docker.com/what-docker.

[19] reTHINK project, "rethink-project/dev-hyperty-toolkit," May 2017. https://github.com/reTHINK-project/dev-hyperty-toolkit/.

[20] "We are smarter together," Aug 2017. http://smarter-together.eu/.

[21] "Our vision," Aug 2017. http://www.grow-smarter.eu/home/.

[22] "Community research and development information service - mysmartlife," Aug 2017. http://cordis.europa.eu/project/rcn/206242_en.html.

[23] K.-D. Borchardt, *The ABC of European Union law*. Publications Office of the European Union, 2010.

[24] C. de la Unión Europea, "Directiva 95/56/ce," 1995. http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:31995L0046&from=en.

[25] C. de la Unión Europea, "Directiva 2002/58/ce," 2002. http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32002L0058&qid=1505063507016&from=EN.

[26] C. de la Unión Europea, "Reglamento 2016/679," 2016. http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32016R0679&qid=1505063507016&from=EN.

[27] C. de la Unión Europea, "Reglamento 283/2014," 2014. http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32014R0283&qid=1505061688464&from=EN.

[28] C. de la Unión Europea, "Reglamento 1083/2012," 2012. http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32012R1083&qid=1505062391386&from=EN.

[29] C. de la Unión Europea, "Directiva 2016/2102," 2016. http://eur-lex.europa.eu/legal-content/ES/TXT/PDF/?uri=CELEX:32016L2102&from=EN.

Summary of Bachelor Thesis

Development and evaluation of real time communication services based on hyperlinked entities

Elias Solana Ortigosa

September 26, 2017

1 Introduction

This Bachelor thesis is based on the work done during my intership in Deutsche Telekom in Berlin, Germany. The main task during those 6 months was to create a web application that developes real time communication systems using the reTHINK concepts.

In the present days, two categories divide the services provided through the Internet: operator-based and web-based. In the first category, the operator provides the services using its own network, meaning strong user privacy and good quality of service. On the other hand, it presents big problems in terms of inter-operability among operators. In the second category, also known as OTT (over-the-top), the lack of an own network means that the aforementioned strengths of the operator-based services are weaknesses for this kind of services. However, web-based services are currently taking the lead in the innovation of new communication platforms.

The reTHINK project is a joint effort done by ten European Telecom companies that aims to create a network arquitecture which bridges the operator-based and web-based services. In this way, operators (especially the smaller ones) would be able to compete with larger web companies, such as Google or Facebook.

In a technical way, the arquitecture developed by reTHINK wants to offer a global and context-based content, assuring not only the inter-operability among providers, but also a good quality of service and strong user privacy performances. These fields are the ones to be covered by the evaluation done in this Bachelor Thesis, through the development of an application.

The application deploys three communication services: videocall, chat and a search engine. The objective of this thesis is to tackle how the perfomances of reTHINK arquitecture are accomplished, evaluating the following technical aspects: re-usability of hyperties as microservices, synchronization mechanism, search engine, 'On-the-fly' protocol and the identity manager.

2 State of the Art

The section named 'State of the Art' makes a summary of the existant techonologies used during the development of the thesis. For this thesis, technologies from different fields are used, mainly web and communication ones, but the most relevant for this study are the technologies developed by the reTHINK project and its architecture.

As mentioned before, reTHINK is a project that aims to provide a framework that overcomes the division of operator-based and web-based services, and this is achieved by creating a new network architecture based on secure peer-to-peer relations among distributed applications centered on the web. This relations are called 'hyperties', a contraction of 'Hyperlinked Entities'.

Internally, the reTHINK project has five strategic objectives: Provide a user-communication framework (Objective 1), design the necessary security and portability to accomplish European regulations (Objective 2), analyze the business impact of the project (Objective 3), validate the reTHINK efectivity in different areas (i.e. real time communications) (Objective 4) and standardize and disseminate the results of the project (Objective 5).

In figure 1 the reTHINK framework is represented, in which the main elements of reTHINK architecture can be seen, such as the hyperties, the Catalogue, the Protostub (highly related to the Protocol 'On-the-fly) and the Hyperty Runtime. It also represents the main mechanisms, like decentralized communication, the data synchronization or the identity management models.

A hyperty is a piece of code written in Javascript that follows the microservices arquitecture philosophym which means that they deploy themselves independently offering different kind of uses, being able to deploy a service when together. For example, a videocall hyperty and a chat hyperty can offer a broader communication service, which is the case of study in this thesis. Hypertys must accomplish the following characteristics: interoperability among them, reliability and re-usability.

The catalogue of reTHINK is a database that contains the necessary objects to make the reTHINK architecture work. The kind of objects that the catalogue can store are the following: hyperty, protocolstub, dataschema, runtime and idp-proxy.

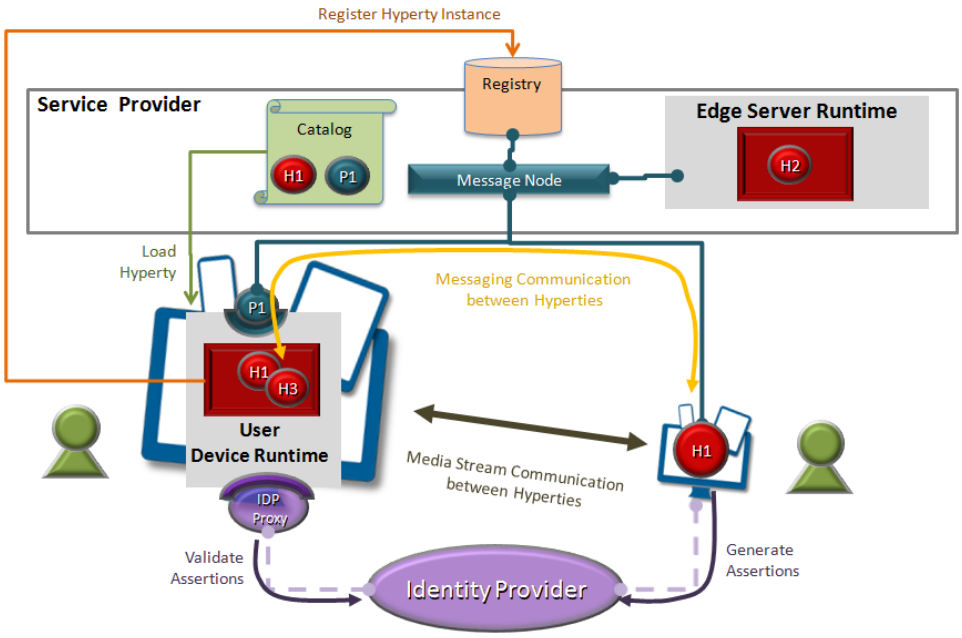


Figure 1: reTHINK Framework. Source: reTHINK project

Protocolstub is a software module that makes a browser understand the protocol of a Message Node of a different provider. This element assures the inter-operability among providers, and follows the 'Protocol On-the-fly' concept, which consists on the dynamic selection and load of the most appropiate protocol for the communication between users.

Hyperty Runtime (defined in the figure as User Device Runtime) is a system installed in every user device which has several modules that interact between them and the outside, allowing the operability with other devices that use hyperties too. The Hyperty Runtime connects with the service providers making use of the protocolstub, connects to the identity provider making use of the IDP proxy, and connects to the Catalogue and the Registry directly.

Regarding the main mechanisms of reTHINK, the de-centralized communication model defines the communication between hyperties in based in objects that perform CRUD operations. The data synchronization model

describes how those objects are updated and how this is notified, and the identity management is based on how the user privacy settings are accomplished, attaching an identity to every single hyperty.

Besides the reTHINK arquitecture, web technologies are widely used along the development of this thesis. HTML, CSS, HTTP/REST and JavaScript are the ones used, being Javascript and JSON the standard used for coding in reTHINK. Also, the standard WebRTC (Web Real-Time Communication), which defines the direct exchange of multimedia data between browsers, is used during the development of the application's videocall system. Lastly, some other technologies like Npm, GIT or Docker are used during the realization of the thesis.

3 Architecture of the development

This section defines the design followed during the development of the aplication. At first, the necessary functionalities of the application are considered: search engine, establish a connection with another user through chat an videocall, and establish a connection in the same way when an invitation from another user is received.

Taking into account these functionalities, a scheme of the working flow of the application is created: down of this, the graphic interface is designed, defining the different blocks needed. This graphic part gets the main focus of this section, since it is a starting point from where to implement all the necessary elements of the reTHINK architecture.

The design of the graphic interface comprehends four main screens: A welcome (introductory) screen, a search and results screen, a chat screen and a videocall plus chat screen. Furthermore, there will be two auxiliary screens that handle a chat invitation and a videocall invitation.

It is also important to note that, to make possible the handling of different conversations with different users, a tab system is created, being able to open, go through and close tabs that are embedded in a tab of the browser. The reason why this system is created instead of using the browser tabs is that, at the moment of the realization of the thesis, deploy a single Hyperty Runtime in different browser tabs was not possible.

When impelementing the reTHINK elements, which is the code that makes the application functional, the first consideration is that two hyperties will be needed: the chat and the videocall hyperties. It is worth to note that, besides the last part of the development in which the videocall hyperty is modified, these hyperties are already created and available in the reTHINK repository. The search engine and the identity manager are also used and implement in the application, requiring minor change in the code.

The last element to be considered in the design of the application is the structure of the code: how it is separated into files and folders. The final decision is to separate the code in a HTML file, which handles the static part of the graphic interface, and three JavaScript files, called videochat, search and tabhandler, which handle the dynamic part of the graphic interface and the code that makes the application functional. There is also a separate folder that contains all the necessary icons and graphic elements.

4 Previous Development

Prior to the development of the application, a previous phase of development is performed to tackle specific characteristics of the reTHINK acrchitecture and get familiar to it. These outcomes are valuable for the development of the application, though any of the code created in this section is used in the application. The aspects to tackle are the data synchronization model, the flexibility of the hyperties when they are modified and the necessary changes in the Hyperty Runtime when building new hyperties.

To achieve it, this previous development is separated in two phases. The first one makes use of the already created Hello World Hyperty, deployed in the Hyperty Toolkit (a tool developed by reTHINK project used to deploy and test hyperties). The data object is modified to test how feasible is to change the format of the data object and which elements are needed to change in the rest of the code of the future applocation.

In the second phase, the Hello World hyperty is taken as a model to develop a new hyperty called Slider, which consists on a slider shown at each user's screen that in which the cursor can be modified in from sides (updating the information at the other side=. This means testing the bi-

directionality of hyperties, and what this implies in the data synchronization system.

5 Development and Validation

The development and validation of the application that this thesis is about takes the longest section of it. On it, we will separate the process in 3 different steps: Create the application, create a communication service for the application, and develop a new videocall hyperty that can handle several calls for the application.

In the first step we will create the application itself. To achieve it, the first need is to configure a development environment that allows to execute all the reTHINK elements (catalogue, registry...). After that, it is necessary to install the Hyperty Runtime in the environment, importing the necessary files to the HTML file.

When the environment is set and ready to use, the next action is to create the tab system. With it, it will be possible to navigate through the different windows of the application, so it will be the first element to be developed. Previous to its implementation in the application, this tab system is created outside, having its graphic elements in a HTML file and the functions that control the interaction in a JavaScript file. When implemented in the application, this code will be held in index.html and search.js respectively.

Having a functional tab system, it is possible to implement the search function in the application. This function will make use of Global Discovery, the search engine developed by reTHINK. This engine will make queries to the registry, and return the user information in a JSON object. The application makes use of this system opening a new tab to perform a search of an user, get its information, present it in the screen and give the possibility to have a chat conversation or make a videocall with it.

The second step when developing the application is to create a communication service for it. This service will be based in two hypertys already created and available in the reTHINK repository, chat and videocall hyperties, and will deploy a communication service in a new tab, creating a new tab for every conversation.

The first step to achieve the deployment of this service is to create the graphic interface. Due to the dynamic nature of the application, these elements will be created during the execution, thus being coded in JavaScript, in the file named videochat.js. This graphic interface has two parts: at the beginning of the connection, the users are having just a chat conversation, but if any of them decides to make a videocall, the chat window will turn into a videocall plus chat window, keeping the history of the chat shown in the previous window. Also, the videocall window must have all the necessary elements for the configuration of the call.

Once the interface is created, it will 'filled with life': all the elements from the reTHINK arquitecture will be implemented on it, using the development environment already set and modifying the code of the application to suit all the reTHINK functionalities. To implement the hyperties in the application, they will be installed in the Hyperty Runtime executed in the application, and the functions provided by them (connect, on notification, accept call) will be called and used through the code of the application. The rest of the elements of the reTHINK arquitecture would be already installed in the development environment.

In the last development phase of the application a new hyperty is created. The reason to create this hyperty is to make the video call fully operational, since the hyperty provided in the reTHINK repository only allows to have one call at the same time. With the development and implementation of this new hyperty in our application, it would be possible to perform several video calls at the same time, each of them in a different tab.

To achieve this goal we will take the video call hyperty as a reference. The first action to take is to separate the hyperty in two different files: the controller, and the hyperty itself. While the application will refer to the hyperty when making use of it, the hyperty itself will not perform any function: this will be performed by the controller. The reason of this separation is that every hyperty will have different controllers under it (stored in a database), each of it taking care of a call with an user. This way one hyperty can handle several calls.

Since this hyperty is created separately, the last stepp will be to implement it in the application. Due to the philospohy of having a single hyperty that can handle several calls, the changes required in the code of the application ar minimal: the application will refer to the hyperty, just adding the

URL of the other user, to let the hyperty address to the correct controller that takes care of that call.

Once the three steps of the application are done, the next procedure is to validate the application to ensure it is fully functional in the real life. Due to logistic difficulties of deploying two versions of our application in two different environments, the application is tested with the previously mentioned Hyperty Toolkit. The results of this validation are satisfactory, working and keeping good performances (quality of video, delay) in all user cases.

6 Conclusions

The last section of this thesis includes an evaluation of both the technical and the general aspects of the reTHINK project and its architecture. The technical aspects to be studied (as mentioned in the first section of this document) are generally met, giving the identity management the best outcomes, and on the other hand being the search engine the one that has more improvement points. However, in a broad point of view, though reTHINK arquitecture accomplishes the objectives for which it was launched, there is a risk of being to late for its explotation.

The reason of it is that web market has changed since the project was launched in 2014, and the big web companies still hold a prevailing position, making difficult for reTHINK to become a standard. Still, reTHINK has potential on areas such as the public sector market, for the evaluation showed good perfomances in identity and user privacy management.

Regarding the future works, by the realization of this Bachelor Thesis, reTHINK concepts were already studied in three different European Union projects related to Smart Cities and citizen participation areas.